

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

Lasse Espeholt^{*1} Hubert Soyer^{*1} Remi Munos^{*1} Karen Simonyan¹ Volodymyr Mnih¹ Tom Ward¹
Yotam Doron¹ Vlad Firoiu¹ Tim Harley¹ Iain Dunning¹ Shane Legg¹ Koray Kavukcuoglu¹

Abstract

In this work we aim to solve a large collection of tasks using a single reinforcement learning agent with a single set of parameters. A key challenge is to handle the increased amount of data and extended training time, which is already a problem in single task learning. We have developed a new distributed agent IMPALA (Importance-Weighted Actor Learner Architecture) that can scale to thousands of machines and achieve a throughput rate of 250,000 frames per second. We achieve stable learning at high throughput by combining decoupled acting and learning with a novel off-policy correction method called V-trace, which was critical for achieving learning stability.

We demonstrate the effectiveness of IMPALA for multi-task reinforcement learning on DMLab-30 (a set of 30 tasks from the DeepMind Lab environment (Beattie et al., 2016)) and Atari-57 (all available Atari games in Arcade Learning Environment (Bellemare et al., 2013a)). Our results show that IMPALA is able to achieve better performance than previous agents, use less data and crucially exhibits positive transfer between tasks as a result of its multi-task approach.

1. Introduction

Deep reinforcement learning methods have recently mastered a wide variety of domains through trial and error learning (Mnih et al., 2015; Silver et al., 2017; 2016; Zoph et al., 2017; Lillicrap et al., 2015; Barth-Maron et al., 2018). While the improvements on tasks like the game of Go (Silver et al., 2017) and Atari games (Horgan et al., 2018) have been dramatic, the progress has been primarily in single task performance, where an agent is trained on each task separately. We are interested in developing new methods

capable of mastering a diverse set of tasks simultaneously as well as environments suitable for evaluating such methods.

One of the main challenges in training a single agent on many tasks at once is scalability. Since the current state-of-the-art methods like A3C (Mnih et al., 2016) or UNREAL (Jaderberg et al., 2017b) can require as much as a billion frames and multiple days to master a single domain, training them on tens of domains at once is too slow to be practical.

We propose the **Importance Weighted Actor-Learner Architecture** (IMPALA) shown in Figure 1. IMPALA is capable of scaling to thousands of machines without sacrificing training stability or data efficiency. Unlike the popular A3C-based agents, in which workers communicate gradients with respect to the parameters of the policy to a central parameter server, IMPALA actors communicate trajectories of experience (sequences of states, actions, and rewards) to a centralised learner. Since the learner in IMPALA has access to full trajectories of experience we use a GPU to perform learning updates on mini-batches of trajectories while aggressively parallelising all non time-dependent operations. This type of decoupled architecture can achieve very high throughput. However, because the policy used to generate a trajectory can lag behind the policy on the learner by several mini-batch updates at the time of gradient calculation, learning becomes off-policy. Therefore, we introduce the *V-trace* off-policy actor-critic algorithm to correct for this harmful discrepancy.

With the scalable architecture and V-trace combined, IMPALA achieves exceptionally high data throughput rates of 250,000 frames per second, making it over 25 times faster than single-machine A3C. Crucially, IMPALA is also more data efficient than A3C based agents and more robust to hyperparameter values and network architectures, allowing it to make better use of deeper neural networks. We demonstrate the effectiveness of IMPALA on multi-task problems using DMLab-30, a new challenge set which consists of 30 diverse cognitive tasks in the 3D DeepMind Lab (Beattie et al., 2016) environment. On this new taskset, IMPALA achieves 49.4% mean capped human normalised score. We further train an agent—one set of weights—to play all games

^{*}Equal contribution ¹DeepMind Technologies, London, United Kingdom. Correspondence to: Lasse Espeholt <lespeholt@google.com>.

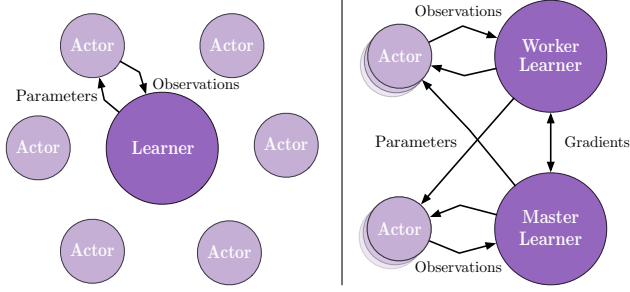


Figure 1. **Left: Single Learner.** Each *actor* generates trajectories and sends them via a queue to the *learner*. Before starting the next trajectory, *actor* retrieves the latest policy parameters from *learner*. **Right: Multiple Synchronous Learners.** Same as *Single Learner*, except the policy parameters are distributed across multiple *learners* that work synchronously.

in the Atari-57 set of tasks, achieving 60% median human normalised score.

2. Related Work

The earliest attempts to scale up deep reinforcement learning relied on using distributed asynchronous SGD (Dean et al., 2012) with multiple workers. Examples include distributed A3C (Mnih et al., 2016), Gorila (Nair et al., 2015) and Ape-X (Horgan et al., 2018) with distributed replay, both distributed versions of Deep Q-Networks (Mnih et al., 2015). Alternatives to asynchronous SGD for RL include using evolutionary processes (Salimans et al., 2017).

More recently, there have been multiple efforts that scale up reinforcement learning further by utilising GPUs. One of the simplest of such methods is batched A2C (Clemente et al., 2017). At every step, batched A2C produces a batch of actions and applies them in parallel to a batch of environments. Therefore, the slowest environment in each batch determines the time it takes to perform the entire batch step (see Figure 2a and 2b). In other words, high variance in environment speed can severely limit performance. Batched A2C works particularly well on Atari environments, because rendering and game logic are computationally very cheap in comparison to the expensive tensor operations performed by reinforcement learning agents. However, more visually or physically complex environments can be slower to render or simulate and can have high variance in the time required for each step. Environments may also have variable length (sub)episodes causing a slowdown when initialising an episode.

The most similar architecture to IMPALA is GA3C (Babaeizadeh et al., 2016), which also uses asynchronous data collection to more effectively utilise GPUs. It decouples the acting/forward pass from the gradient calculation/backward pass by using dynamic batching with a pre-

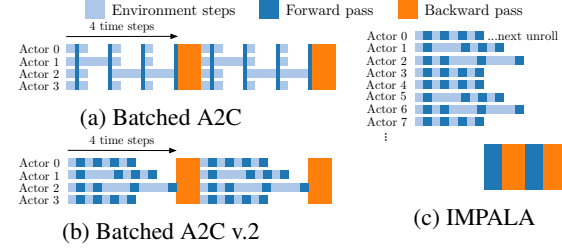


Figure 2. Time line for one unroll with 4 steps using different architectures. The strategies shown in (a) and (b) can lead to low GPU utilisation due to unequal environment rendering time across samples in a single batch. In (a), the actors are synchronised after every step, whereas in (b), after every n steps. IMPALA (c) decouples acting from learning to achieve maximum throughput. Asynchronous actors collect trajectories of experience and send it to a centralised learner.

diction queue. This actor/learner asynchrony in GA3C leads to instabilities during learning, which (Babaeizadeh et al., 2016) only partially mitigates by adding a small constant to action probabilities during the estimation of the policy gradient. In contrast, IMPALA uses the more principled V-trace algorithm.

Related previous work on off-policy RL include (Precup et al., 2000; 2001; Wawrzynski, 2009; Geist & Scherrer, 2014) and (Harutyunyan et al., 2016). The closest work to ours is the Retrace algorithm (Munos et al., 2016) which introduced an off-policy correction for multi-step RL, and has been used in several agent architectures, such as ACER (Wang et al., 2017) and REACTOR (Gruslys et al., 2018). Retrace requires learning state-action-value functions Q in order to make the off-policy correction. However, many actor-critic methods such as (Mnih et al., 2016) learn a state-value function V instead of a state-action-value function Q . In this work we use the state-value function based V-trace, an off-policy correction method for actor-critic.

3. IMPALA

IMPALA (Figure 1) uses an actor-critic setup to learn a policy π and a baseline function V^π . Crucially, in IMPALA the process of generating experiences is decoupled from learning the parameters of π and V^π . The architecture consists of a set of actors, repeatedly generating trajectories of experience, and one or more learners that use the experiences sent from actors to learn π off-policy.

At the beginning of each trajectory, an actor updates its own local policy μ to the latest learner policy π and runs it for n steps in its environment. After n steps, the actor sends the trajectory of states, actions and rewards $x_1, a_1, r_1, \dots, x_n, a_n, r_n$ together with the corresponding policy distributions $\mu(a_t|x_t)$ and initial LSTM state to the

learner through a queue. The learner then continuously updates its policy π on batches of trajectories, each collected from many actors. This simple architecture enables the learner(s) to be accelerated using GPUs and actors to easily be distributed across many machines. However, the learner policy π is potentially several SGD updates ahead of the actor’s policy μ at the time of update, therefore there is a *policy-lag* between the actors and learner(s).

V-trace compensates for the lag and improves stability and convergence of learning, allowing IMPALA to achieve extremely high data throughput whilst maintaining learning efficiency. Importantly, not all actors need to be online at the same time as the experiences are queued, adding redundancy and fault tolerance to the distributed model. The communication bandwidth required for IMPALA is often lower than for distributed A3C variants since in IMPALA the actors send observations rather than gradients, which in practice are larger than the observations.

With the introduction of very deep model architectures, the speed of a single GPU is often the limiting factor during training. IMPALA can be used with a learner that is distributed across multiple GPUs to train large neural networks efficiently as shown in Figure 1. The parameters are distributed across the learner machines and each actor retrieves the model parameters from all the learners in parallel. Each actor sends observations directly to one of the learners. IMPALA performs synchronised parameter updates even in this distributed learner setup because it is vital to maintaining data efficiency when scaling to more machines (Chen et al., 2016).

3.1. Efficiency Optimisations

GPUs and many-core CPUs benefit greatly from running few large, parallelisable operations instead of many small operations. Since the learner in IMPALA performs updates on entire batches of trajectories, it is able to parallelise more of its computations than an online agent like A3C, which interacts with the environment step-by-step (see Figure 3a). As an example, a typical deep RL agent features a convolutional network followed by an Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and a fully connected output layer after the LSTM. An IMPALA learner applies the convolutional network to all inputs in parallel by folding the time dimension into the batch dimension (see Figure 3b). Similarly, it also applies the output layer to all time steps in parallel once the LSTM states for all time steps are computed. This optimisation increases the effective batch size to thousands and allows for each item in the batch to be run in parallel instead of operating step by step.

For LSTM-based agents, we also obtain significant speedups on the learner by exploiting the network structure and op-

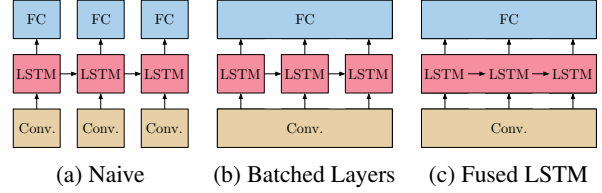


Figure 3. Optimisations to increase the effective batch size and GPU utilisation. **a)** A naive, but straightforward implementation without any batching in convolutional or LSTM layers. **b)** Batching applied on the input convolutional and output layers. **c)** Batching is applied across the whole network with fused LSTM operations.

eration fusion for LSTMs (Appleyard et al., 2016) (see Figure 3c).

Finally, we also make use of several off the shelf optimisations available in TensorFlow (Abadi et al., 2017) such as preparing the next batch of data for the learner while still performing computation, compiling parts of the computational graph with XLA (a TensorFlow Just-In-Time compiler) and optimising the data format to get the maximum performance from the cuDNN framework (Chetlur et al., 2014).

4. V-trace

Off-policy learning is important in this decoupled distributed actor-learner architecture because of the lag between when actions are generated by the actors and when the learner estimates the gradient. To this end, we introduce a novel off-policy actor-critic algorithm for the learner, called V-trace.

First, let us introduce some notations. We consider the problem of discounted infinite-horizon RL in Markov Decision Processes (MDP), see (Puterman, 1994; Sutton & Barto, 1998) where the goal is to find a policy π that maximises the expected sum of future discounted rewards: $V^\mu(x) \stackrel{\text{def}}{=} \mathbb{E}_\mu \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$, where $\gamma \in [0, 1)$ is the discount factor, $r_t = r(x_t, a_t)$ is the reward at time t , x_t is the state at time t (initialised in $x_0 = x$) and $a_t \sim \mu(\cdot | x_t)$ is the action generated by following some policy μ .

The goal of an off-policy RL algorithm is to use trajectories generated by some policy μ , called the *behaviour policy*, to learn the value function V^π of another policy π (possibly different from μ), called the *target policy*.

4.1. V-trace target

Consider a trajectory $(x_t, a_t, r_t)_{t=s}^{t=s+n}$ generated by the actor following some policy μ . We define the n -steps V-trace target for $V(x_s)$, our value approximation at state x_s , as:

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

where $\delta_t V \stackrel{\text{def}}{=} \rho_t (r_t + \gamma V(x_{t+1}) - V(x_t))$ is a temporal difference for V , and $\rho_t \stackrel{\text{def}}{=} \min(\bar{\rho}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)})$ and $c_i \stackrel{\text{def}}{=} \min(\bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)})$ are truncated importance sampling (IS) weights (we make use of the notation $\prod_{i=s}^{t-1} c_i = 1$ for $s = t$). In addition we assume that the truncation levels are such that $\bar{\rho} \geq \bar{c}$.

Notice that in the on-policy case (when $\pi = \mu$), and assuming that $\bar{c} \geq 1$, then all $c_i = 1$ and $\rho_t = 1$, thus (1) rewrites

$$\begin{aligned} v_s &= V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} (r_t + \gamma V(x_{t+1}) - V(x_t)) \\ &= \sum_{t=s}^{s+n-1} \gamma^{t-s} r_t + \gamma^n V(x_{s+n}), \end{aligned} \quad (2)$$

which is the on-policy n -steps Bellman target. Thus in the on-policy case, V-trace reduces to the on-policy n -steps Bellman update. This property (which Retrace (Munos et al., 2016) does not have) allows one to use the same algorithm for off- and on-policy data.

Notice that the (truncated) IS weights c_i and ρ_t play different roles. The weight ρ_t appears in the definition of the temporal difference $\delta_t V$ and defines the fixed point of this update rule. In a tabular case, where functions can be perfectly represented, the fixed point of this update (i.e., when $V(x_s) = v_s$ for all states), characterised by $\delta_t V$ being equal to zero in expectation (under μ), is the value function $V^{\pi_{\bar{\rho}}}$ of some policy $\pi_{\bar{\rho}}$, defined by

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

(see the analysis in Appendix A). So when $\bar{\rho}$ is infinite (i.e. no truncation of ρ_t), then this is the value function V^π of the target policy. However if we choose a truncation level $\bar{\rho} < \infty$, our fixed point is the value function $V^{\pi_{\bar{\rho}}}$ of a policy $\pi_{\bar{\rho}}$ which is somewhere between μ and π . At the limit when $\bar{\rho}$ is close to zero, we obtain the value function of the behaviour policy V^μ . In Appendix A we prove the contraction of a related V-trace operator and the convergence of the corresponding online V-trace algorithm.

The weights c_i are similar to the ‘‘trace cutting’’ coefficients in Retrace. Their product $c_s \dots c_{t-1}$ measures how much a temporal difference $\delta_t V$ observed at time t impacts the update of the value function at a previous time s . The more dissimilar π and μ are (the more off-policy we are), the larger the variance of this product. We use the truncation level \bar{c} as a variance reduction technique. However notice that this truncation does not impact the solution to which we converge (which is characterised by $\bar{\rho}$ only).

Thus we see that the truncation levels \bar{c} and $\bar{\rho}$ represent different features of the algorithm: $\bar{\rho}$ impacts the nature of the value function we converge to, whereas \bar{c} impacts the speed at which we converge to this function.

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

4.2. Actor-Critic algorithm

POLICY GRADIENT

In the on-policy case, the gradient of the value function $V^\mu(x_0)$ with respect to some parameter of the policy μ 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 $Q^\mu(x_s, a_s) \stackrel{\text{def}}{=} \mathbb{E}_\mu [\sum_{t \geq s} \gamma^{t-s} r_t | x_s, a_s]$ is the state-action value of policy μ at (x_s, a_s) . This is usually implemented by a stochastic gradient ascent that updates the policy parameters in the direction of $\mathbb{E}_{a_s \sim \mu(\cdot|x_s)} [\nabla \log \mu(a_s|x_s) q_s | x_s]$, where q_s is an estimate of $Q^\mu(x_s, a_s)$, and averaged over the set of states x_s that are visited under some behaviour policy μ .

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

$$\mathbb{E}_{a_s \sim \mu(\cdot|x_s)} \left[\frac{\pi_{\bar{\rho}}(a_s|x_s)}{\mu(a_s|x_s)} \nabla \log \pi_{\bar{\rho}}(a_s|x_s) q_s | x_s \right] \quad (4)$$

where $q_s \stackrel{\text{def}}{=} r_s + \gamma v_{s+1}$ is an estimate of $Q^{\pi_{\bar{\rho}}}(x_s, a_s)$ built from the V-trace estimate v_{s+1} at the next state x_{s+1} . The reason why we use q_s instead of v_s as the target for our Q-value $Q^{\pi_{\bar{\rho}}}(x_s, a_s)$ is that, assuming our value estimate is correct at all states, i.e. $V = V^{\pi_{\bar{\rho}}}$, then we have $\mathbb{E}[q_s|x_s, a_s] = Q^{\pi_{\bar{\rho}}}(x_s, a_s)$ (whereas we do not have this property if we choose $q_t = v_t$). See Appendix A.

In order to reduce the variance of the policy gradient estimate (4), we usually subtract from q_s a state-dependent baseline, such as the current value approximation $V(x_s)$.

Finally notice that (4) 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 $Q^\pi(x_s, a_s)$. Taking into account these remarks, we derive the following canonical V-trace actor-critic algorithm.

V-TRACE ACTOR-CRITIC ALGORITHM

Consider a parametric representation V_θ of the value function and the current policy π_ω . Trajectories have been gen-

erated by actors following some behaviour policy μ . The V-trace targets v_s are defined by (1). At training time s , the value parameters θ are updated by gradient descent on the l_2 loss to the target v_s , i.e., in the direction of

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

and the policy parameters ω in the direction of the policy gradient:

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

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 hyperparameters of the algorithm.

5. Experiments

We investigate the performance of IMPALA under multiple settings. For data efficiency, computational performance and effectiveness of the off-policy correction we look at the learning behaviour of IMPALA agents trained on individual tasks. For multi-task learning we focus on training one agent—one set of weights—on a newly introduced collection of 30 DeepMind Lab tasks and on all 57 games of the Atari Learning Environment (Bellemare et al., 2013a).

For all the experiments we have used two different model architectures: a shallow model similar to (Mnih et al., 2016) with an LSTM before the policy and value (shown in Figure 4 (left)) and a deeper residual model (He et al., 2016) (shown in Figure 4 (right)). For tasks with a language channel we used an LSTM with text embeddings as input.

5.1. Computational Performance

High throughput, computational efficiency and scalability are among the main design goals of IMPALA. To demonstrate that IMPALA outperforms current algorithms in these metrics we compare A3C (Mnih et al., 2016), batched A2C variations and IMPALA variants with various optimisations. Table 1 details the results for single-machine and multi-machine versions of these variants on two DeepMind Lab tasks (*seekavoid_arena_01* and *rooms_keys_doors_puzzle*). In the single-machine case, IMPALA achieves the top performance on both tasks, ahead of all batched A2C variants and ahead of A3C. However, the distributed, multi-machine setup is where IMPALA can really demonstrate its scalability. Together with applying the optimisations from Section 3.1 to speed up the

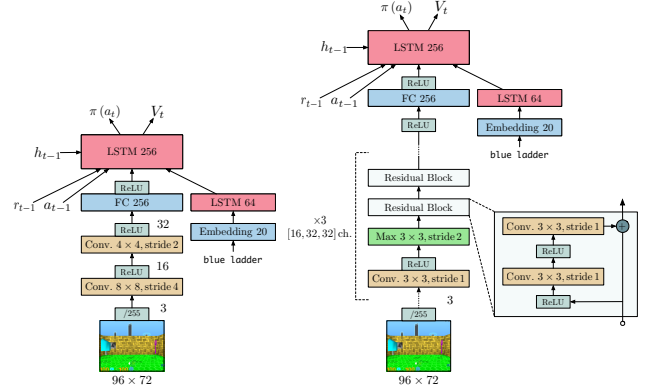


Figure 4. Model Architectures. **Left:** Small architecture used for training on individual levels, containing 2 convolutional layers and 1.2 million parameters. **Right:** Large architecture used for training on DMLab-30 multi-task challenge, containing 15 convolutional layers and 1.6 million parameters.

GPU-based learner—and given enough CPU-based actors—the IMPALA agent achieves a throughput rate of 250,000 frames/sec or 21 billion frames/day. To reduce the number of actors needed per learner, one can use auxiliary losses, data from experience replay or other expensive learner-only computation.

5.2. Single-Task Training

To investigate IMPALA’s learning dynamics, we employ the single-task scenario where we train agents individually on 5 different DeepMind Lab tasks. The task set consists of a planning task, two maze navigation tasks, a laser tag task with scripted bots and a simple fruit collection task.

We perform hyperparameter sweeps over the weighting of *entropy regularisation*, the *learning rate* and the *RMSProp epsilon*. For each experiment we use an identical set of 24 pre-sampled hyperparameter combinations from the ranges in Appendix C.1. The other hyperparameters were fixed to values specified in Appendix C.3.

5.2.1. CONVERGENCE AND STABILITY

Figure 5 shows a comparison between IMPALA, A3C and batched A2C. In all of the 5 tasks, either batched A2C or IMPALA reach the best final average return and in all tasks but *seekavoid_arena_01* they are ahead of A3C throughout the entire course of training. IMPALA outperforms the synchronous batched A2C on 2 out of 5 tasks while achieving much higher throughput (see Table 1). We hypothesise that this behaviour could stem from the V-trace off-policy correction acting similarly to generalised advantage estimation (Schulman et al., 2016) and asynchronous data collection yielding more diverse batches of experience.

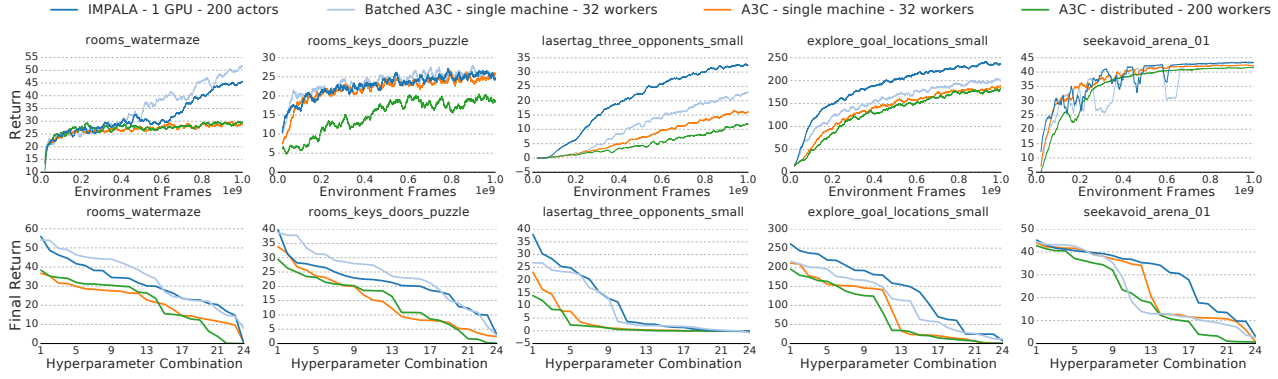


Figure 5. **Top Row:** Comparisons between IMPALA and A3C for single task training on 5 DeepMind Lab tasks. Each curve is obtained by taking the mean of the best 3 curves based on final return. IMPALA generally achieves better performance compared to A3C. **Bottom Row:** Stability across hyperparameter combinations obtained by sorting the final performance across different hyperparameter combinations from high to low. IMPALA is consistently more stable than A3C.

Architecture	CPU	GPU	FPS	
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 ¹	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 Limited by amount of rendering possible on a single machine.

Table 1. Throughput on two tasks (seekavoid.arena.01 (task 1) and rooms.keys.doors.puzzle (task 2) where the latter has variable length episodes and relatively slow restarts) using the model in Figure 4. Listed in frames per second (4 times the agent steps due to action repeat). Batched A2C and IMPALA uses batch size 32 if not otherwise mentioned.

In addition to reaching better final performance, IMPALA is also more robust to the choice of hyperparameters than A3C. Figure 5 compares the final performance of the aforementioned methods across different hyperparameter combinations, sorted by average final return from high to low. Note that IMPALA achieves higher scores over a larger number of combinations than A3C.

5.2.2. V-TRACE ANALYSIS

To analyse V-trace we investigate four different algorithms:

- 1. No-correction** - No off-policy correction.
- 2. ϵ -correction** - Add a small value ($\epsilon = 1e-6$) during

	Task 1	Task 2	Task 3	Task 4	Task 5
Without Replay					
V-trace	46.8	32.9	31.3	229.2	43.8
1-Step	51.8	35.9	25.4	215.8	43.7
ϵ -correction	44.2	27.3	4.3	107.7	41.5
No-correction	40.3	29.1	5.0	94.9	16.1
With Replay					
V-trace	47.1	35.8	34.5	250.8	46.9
1-Step	54.7	34.4	26.4	204.8	41.6
ϵ -correction	30.4	30.2	3.9	101.5	37.6
No-correction	35.0	21.1	2.8	85.0	11.2

Tasks: rooms.watermaze, rooms.keys.doors.puzzle, lasertag.three.opponents.small, explore.goal.locations.small, seekavoid.arena.01

Table 2. Average final return over 3 best hyperparameters for different off-policy correction methods on 5 DeepMind Lab tasks. When the lag in policy is negligible both V-trace and 1-step importance sampling perform similarly well and better than ϵ -correction and having No-correction. However, when the lag increases due to use of experience replay, V-trace performs better than all other methods in 4 out of 5 tasks.

gradient calculation to prevent $\log \pi(a)$ from becoming very small and leading to numerical instabilities, similar to (Babaeizadeh et al., 2016).

3. 1-step importance sampling - No off-policy correction when optimising $V(x)$. For the policy gradient, multiply the advantage at each time step by the corresponding importance weight. This variant is similar to V-trace without “traces” and is included to investigate the importance of “traces” in V-trace.

4. V-trace as described in Section 4.

For V-trace and 1-step importance sampling we clip each importance weight ρ_t and c_t at 1 (i.e. $\bar{c} = \bar{\rho} = 1$). This

Model	Test score
A3C, deep	23.8%
IMPALA, shallow	37.1%
IMPALA-Experts, deep	44.5%
IMPALA, deep	46.5%
IMPALA, deep, PBT	49.4%
IMPALA, deep, PBT, 8 learners	49.1%

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.

reduces the variance of the gradient estimate but introduces a bias. Out of $\bar{\rho} \in [1, 10, 100]$ we found that $\bar{\rho} = 1$ worked best.

We evaluate all algorithms on the set of 5 DeepMind Lab tasks from the previous section. We also add an experience replay buffer on the learner to increase the off-policy gap between π and μ . In the experience replay experiments we draw 50% of the items in each batch uniformly at random from the replay buffer. Table 2 shows the final performance for each algorithm with and without replay respectively. In the no replay setting, V-trace performs best on 3 out of 5 tasks, followed by 1-step importance sampling, ϵ -correction and No-correction. Although 1-step importance sampling performs similarly to V-trace in the no-replay setting, the gap widens on 4 out of 5 tasks when using experience replay. This suggests that the cruder 1-step importance sampling approximation becomes insufficient as the target and behaviour policies deviate from each other more strongly. ϵ -correction improves significantly over No-correction on two tasks but lies far behind the importance-sampling based methods, particularly in the more off-policy setting with experience replay. Figure D.1 shows results of a more detailed analysis. Figure D.2 shows that the importance-sampling based methods also perform better across all hyperparameters and are typically more robust.

5.3. Multi-Task Training

IMPALA’s high data throughput and data efficiency allow us to train not only on one task but on multiple tasks in parallel with only a minimal change to the training setup. Instead of running the same task on all actors, we allocate a fixed number of actors to each task in the multi-task suite.

5.3.1. DMLAB-30

To test IMPALA’s performance in a multi-task setting we use DMLab-30, a set of 30 diverse tasks built using the DeepMind Lab environment. Among the many task types in the suite are visually complex environments with natural-looking terrain, instruction-based tasks with grounded language, navigation tasks, cognitive (Leibo et al.,

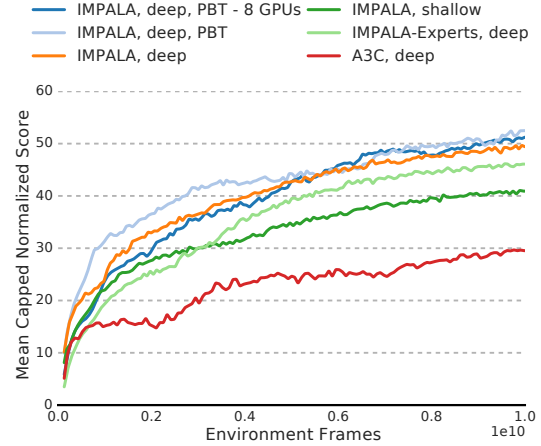


Figure 6. Performance of best agent in each sweep/population during training on the DMLab-30 task-set wrt. environment data consumed. IMPALA with multi-task training is not only faster, it also converges at higher accuracy with better data efficiency across all 30 tasks. The x-axis is data consumed by one agent out of a hyperparameter sweep/PBT population of 24 agents, total data consumed across the whole population/sweep can be obtained by multiplying with the population/sweep size.

2018) and first-person tagging tasks featuring scripted bots as opponents. A detailed description of DMLab-30 and the tasks are available at github.com/deepmind/lab and deepmind.com/dm-lab-30.

We compare multiple variants of IMPALA with a distributed A3C implementation. Except for agents using population-based training (PBT) (Jaderberg et al., 2017a), all agents are trained with hyperparameter sweeps across the same range given in Appendix C.1. We report mean capped human normalised score where the score for each task is capped at 100% (see Appendix B). Using mean capped human normalised score emphasises the need to solve multiple tasks instead of focusing on becoming super human on a single task. For PBT we use the mean capped human normalised score as fitness function and tune entropy cost, learning rate and RMSProp ϵ . See Appendix E for the specifics of the PBT setup.

In particular, we compare the following agent variants. *A3C, deep*, a distributed implementation with 210 workers (7 per task) featuring the deep residual network architecture (Figure 4 (Right)). *IMPALA, shallow* with 210 actors and *IMPALA, deep* with 150 actors both with a single learner. *IMPALA, deep, PBT*, the same as *IMPALA, deep*, but additionally using the PBT method (Jaderberg et al., 2017a) for hyperparameter optimisation. Finally *IMPALA, deep, PBT, 8 learners*, which utilises 8 learner GPUs to maximise learning speed. We also train IMPALA agents in an expert setting, *IMPALA-Experts, deep*, where a separate agent is trained per task. In this case we did *not* optimise hyperparameters

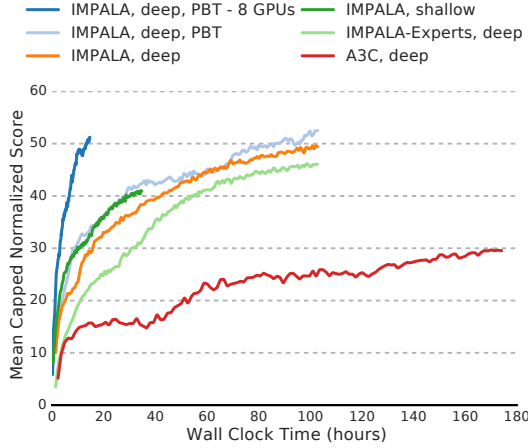


Figure 7. Performance on the DMLab-30 task-set wrt. wall-clock time. All models used the deep architecture (Figure 4). The high throughput rate of IMPALA results in orders of magnitude faster learning

for each task and instead used the same hyperparameters for all 30 expert agents.

Table 3 and Figure 6 show all variants of IMPALA performing much better than the deep distributed A3C. Moreover, the deep variant of IMPALA performs better than the shallow network version not only in terms of final performance but throughout the entire training with better data efficiency. Note in Table 3 that *IMPALA, deep, PBT, 8 learners*, although providing much higher throughput, reaches the same final performance as the 1 GPU *IMPALA, deep, PBT* in the same number of steps, i.e. with no loss in data efficiency. Of particular importance is the gap between the *IMPALA-Experts* which were trained on each task individually and *IMPALA, deep, PBT* which was trained on all tasks at once. As Figure 6 shows, the multi-task version is above *IMPALA-Experts* throughout training which suggests that the multi-task agent benefits from a large amount of positive transfer between tasks.

Comparing A3C to IMPALA with respect to wall clock time (Figure 7) further highlights the scalability gap between the two approaches. IMPALA with 1 learner takes only around 10 hours to reach the same performance that A3C approaches after 7.5 days. Using 8 learner GPUs instead of 1 further speeds up training by a factor of 7.

5.3.2. ATARI

The Atari Learning Environment (ALE) (Bellemare et al., 2013b) has been the testing ground of most recent deep reinforcement agents. Its 57 tasks pose challenging reinforcement learning problems including exploration, planning, reactive play and complex visual input. Most games feature very different visuals and game mechanics which

makes this domain particularly challenging for multi-task learning.

We train IMPALA and A3C agents on each game individually and compare their performance using the deep network (without the LSTM) introduced in Section 5. We also provide results using a shallow network that is equivalent to the feed forward network used in (Mnih et al., 2016) which features a three-layer convolutional. The network is provided with a short term history by stacking the 4 most recent observations at each step. For details on pre-processing and hyperparameter setup please refer to Appendix F.

In addition to individual per-game experts, trained for 200 million frames, we train an IMPALA Atari-57 agent—one agent, one set of weights—on all 57 Atari games at once for 200 million frames per game or a total of 11.4 billion frames. We use population based training with a population size of 24 to adopt entropy regularisation, learning rate, RMSProp ϵ and the global gradient norm clipping threshold throughout training.

Human Normalised Return	Median	Mean
A3C, shallow, experts	54.9%	285.9%
A3C, deep, experts	117.9%	503.6%
IMPALA, shallow, experts	93.2%	466.4%
IMPALA, deep, experts	191.8%	957.6%
IMPALA, deep, multi-task	59.7%	176.9%

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

We compare all algorithms in terms of median human normalised score across all 57 Atari games. Evaluation follows a standard protocol, each game-score is the mean over 200 evaluation episodes, each episode was started with a random number of no-op actions (uniformly chosen from [1, 30]) to combat the determinism of the ALE environment.

As table 4 shows, IMPALA experts provide both better final performance and data efficiency than their A3C counterparts in the deep and the shallow configuration. As in our DeepMind Lab experiments, the deep residual network leads to higher scores than the shallow network, irrespective of the reinforcement learning algorithm used. Note that the shallow IMPALA experiment completes training over 200 millions frames in less than one hour.

We want to particularly emphasise that *IMPALA, deep, multi-task*, a single agent trained on all 57 ALE games at once, reaches 59.7% median human normalised score. Despite the high diversity in visual appearance and game mechanics within the ALE suite, IMPALA multi-task still manages to stay competitive to *A3C, shallow, experts*, commonly used as a baseline in related work. ALE is typically considered a hard multi-task environment, often accompanied

by negative transfer between tasks (Rusu et al., 2016). To our knowledge, IMPALA is the first agent to be trained in a multi-task setting on all 57 games of ALE that is competitive with a standard expert baseline.

6. Conclusion

We have introduced a new highly scalable distributed agent, IMPALA, and a new off-policy learning algorithm, V-trace. We have shown that IMPALA can achieve much higher data throughput (250K frames/sec vs. 50K frames/sec) due to its simple but scalable distributed architecture. The ability to train agents at this scale directly translates to very quick turnaround for investigating new ideas and opens up unexplored opportunities.

V-trace is a general off-policy learning algorithm that is more stable and robust compared to other off-policy correction methods for actor critic agents. Crucially, we have demonstrated that IMPALA achieves better performance compared to A3C variants in terms of data efficiency, stability and final performance. We have further evaluated IMPALA on the new DMLab-30 set and the Atari-57 set. To the best of our knowledge, IMPALA is the first Deep-RL agent that has been successfully tested in such large-scale multi-task settings and it has shown superior performance compared to A3C based agents (49.4% vs. 23.8% human normalised score). Most importantly, our experiments on DMLab-30 show that, in the multi-task setting, positive transfer between individual tasks lead IMPALA to achieve better performance compared to the expert training setting. We believe that IMPALA provides a very simple yet scalable and robust framework for building better Deep-RL agents and has the potential to enable research on new challenges.

Acknowledgements

We would like to thank Denis Teplyashin, Ricardo Barreira, Manuel Sanchez for their work improving the performance on DMLab-30 environments and Matteo Hessel, Jony Hudson, Igor Babuschkin, Max Jaderberg, Ivo Danihelka and David Silver for their comments and insightful discussions.

References

- Abadi, Martin, Isard, Michael, and Murray, Derek G. A computational model for tensorflow: An introduction. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2017, 2017. ISBN 978-1-4503-5071-6.
- Appleyard, Jeremy, Kociský, Tomás, and Blunsom, Phil. Optimizing performance of recurrent neural networks on gpus. *CoRR*, abs/1604.01946, 2016.
- Babaeizadeh, Mohammad, Frosio, Iuri, Tyree, Stephen, Clemons, Jason, and Kautz, Jan. GA3C: GPU-based A3C for deep reinforcement learning. *NIPS Workshop*, 2016.
- Barth-Maroon, Gabriel, Hoffman, Matthew W., Budden, David, Dabney, Will, Horgan, Dan, Tirumala, Dhruva, Muldal, Alistair, Heess, Nicolas, and Lillicrap, Timothy. Distributional policy gradients. *International Conference on Learning Representations*, 2018.
- Beattie, Charles, Leibo, Joel Z., Teplyashin, Denis, Ward, Tom, Wainwright, Marcus, Kuttler, Heinrich, Lefrancq, Andrew, Green, Simon, Valdes, Victor, Sadik, Amir, Schrittwieser, Julian, Anderson, Keith, York, Sarah, Cant, Max, Cain, Adam, Bolton, Adrian, Gaffney, Stephen, King, Helen, Hassabis, Demis, Legg, Shane, and Petersen, Stig. Deepmind lab. *CoRR*, abs/1612.03801, 2016.
- Bellemare, Marc G., Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013a.
- Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013b.
- Chen, Jianmin, Monga, Rajat, Bengio, Samy, and Józefowicz, Rafal. Revisiting distributed synchronous SGD. *CoRR*, abs/1604.00981, 2016.
- Chetlur, Sharan, Woolley, Cliff, Vandermersch, Philippe, Cohen, Jonathan, Tran, John, Catanzaro, Bryan, and Shohler, Evan. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.
- Clemente, Alfredo V., Martínez, Humberto Nicolás Castejón, and Chandra, Arjun. Efficient parallel methods for deep reinforcement learning. *CoRR*, abs/1705.04862, 2017.
- Dean, Jeffrey, Corrado, Greg, Monga, Rajat, Chen, Kai, Devin, Matthieu, Mao, Mark, Ranzato, Marc’Aurelio, Senior, Andrew, Tucker, Paul, Yang, Ke, Le, Quoc V., and Ng, Andrew Y. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*, pp. 1223–1231, 2012.
- Geist, Matthieu and Scherrer, Bruno. Off-policy learning with eligibility traces: A survey. *The Journal of Machine Learning Research*, 15(1):289–333, 2014.
- Gruslys, Audrunas, Dabney, Will, Azar, Mohammad Gheshlaghi, Piot, Bilal, Bellemare, Marc G., and Munos, Rémi. The Reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. *International Conference on Learning Representations*, 2018.

- Harutyunyan, Anna, Bellemare, Marc G., Stepleton, Tom, and Munos, Rémi. *Q(λ) with Off-Policy Corrections*, pp. 305–320. Springer International Publishing, Cham, 2016.
- He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016.
- Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Horgan, Dan, Quan, John, Budden, David, Barth-Maron, Gabriel, Hessel, Matteo, van Hasselt, Hado, and Silver, David. Distributed prioritized experience replay. *International Conference on Learning Representations*, 2018.
- Jaderberg, Max, Dalibard, Valentin, Osindero, Simon, Czarnecki, Wojciech M., Donahue, Jeff, Razavi, Ali, Vinyals, Oriol, Green, Tim, Dunning, Iain, Simonyan, Karen, Fernando, Chrisantha, and Kavukcuoglu, Koray. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017a.
- Jaderberg, Max, Mnih, Volodymyr, Czarnecki, Wojciech Marian, Schaul, Tom, Leibo, Joel Z, Silver, David, and Kavukcuoglu, Koray. Reinforcement learning with unsupervised auxiliary tasks. *International Conference on Learning Representations*, 2017b.
- Leibo, Joel Z, d’Autume, Cyprien de Masson, Zoran, Daniel, Amos, David, Beattie, Charles, Anderson, Keith, Castañeda, Antonio García, Sanchez, Manuel, Green, Simon, Gruslys, Audrunas, et al. Psychlab: A psychology laboratory for deep reinforcement learning agents. *arXiv preprint arXiv:1801.08116*, 2018.
- Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- Mnih, Volodymyr, Badia, Adrià Puigdomènech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P., Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016.
- Munos, Rémi, Stepleton, Tom, Harutyunyan, Anna, and Bellemare, Marc. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1046–1054, 2016.
- Nair, Arun, Srinivasan, Praveen, Blackwell, Sam, Alciçek, Cagdas, Fearon, Rory, Maria, Alessandro De, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, Legg, Shane, Mnih, Volodymyr, Kavukcuoglu, Koray, and Silver, David. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015.
- Precup, Doina, Sutton, Richard S, and Singh, Satinder. Eligibility traces for off-policy policy evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.
- Precup, Doina, Sutton, Richard S., and Dasgupta, Sanjoy. Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning*, pp. 417–424, 2001.
- Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.
- Rusu, Andrei A, Rabinowitz, Neil C, Desjardins, Guillaume, Soyer, Hubert, Kirkpatrick, James, Kavukcuoglu, Koray, Pascanu, Razvan, and Hadsell, Raia. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.
- Salimans, Tim, Ho, Jonathan, Chen, Xi, and Sutskever, Ilya. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.
- Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael, and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- Silver, David, Huang, Aja, Maddison, Christopher J., Guez, Arthur, Sifre, Laurent, van den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, Dieleman, Sander, Grewe, Dominik, Nham, John, Kalchbrenner, Nal, Sutskever, Ilya, Lillicrap, Timothy, Leach, Madeleine, Kavukcuoglu, Koray, Graepel, Thore, and Hassabis, Demis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.
- Silver, David, Schrittwieser, Julian, Simonyan, Karen, Antonoglou, Ioannis, Huang, Aja, Guez, Arthur, Hubert, Thomas, Baker, Lucas, Lai, Matthew, Bolton, Adrian,

Chen, Yutian, Lillicrap, Timothy, Hui, Fan, Sifre, Laurent, Driessche, George van den, Graepel, Thore, and Hassabis, Demis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 10 2017. ISSN 0028-0836. doi: 10.1038/nature24270.

Sutton, R.S. and Barto, A.G. *Reinforcement learning: An introduction*, volume 116. Cambridge Univ Press, 1998.

Wang, Ziyu, Bapst, Victor, Heess, Nicolas, Mnih, Volodymyr, Munos, Remi, Kavukcuoglu, Koray, and de Freitas, Nando. Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations*, 2017.

Wawrzynski, Pawel. Real-time reinforcement learning by sequential actor-critics and experience replay. *Neural Networks*, 22(10):1484–1497, 2009.

Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.

Supplementary Material

A. Analysis of V-trace

A.1. V-trace operator

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

$$\mathcal{R}V(x) \stackrel{\text{def}}{=} V(x) + \mathbb{E}_\mu \left[\sum_{t \geq 0} \gamma^t (c_0 \dots c_{t-1}) \rho_t (r_t + \gamma V(x_{t+1}) - V(x_t)) \mid x_0 = x, \mu \right], \quad (5)$$

where the expectation \mathbb{E}_μ is with respect to the policy μ which has generated the trajectory $(x_t)_{t \geq 0}$, i.e., $x_0 = x$, $x_{t+1} \sim p(\cdot | x_t, a_t)$, $a_t \sim \mu(\cdot | x_t)$. Here we consider the infinite-horizon operator but very similar results hold for the n -step truncated operator.

Theorem 1. Let $\rho_t = \min(\bar{\rho}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)})$ and $c_t = \min(\bar{c}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)})$ 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 (5) has a unique fixed point $V^{\pi_{\bar{\rho}}}$, which is the value function of the policy $\pi_{\bar{\rho}}$ defined by

$$\pi_{\bar{\rho}}(a|x) \stackrel{\text{def}}{=} \frac{\min(\bar{\rho}\mu(a|x), \pi(a|x))}{\sum_{b \in A} \min(\bar{\rho}\mu(b|x), \pi(b|x))}, \quad (6)$$

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

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

Remark 3. The truncation levels \bar{c} and $\bar{\rho}$ play different roles in this operator:

- $\bar{\rho}$ impacts the fixed-point of the operator, thus the policy $\pi_{\bar{\rho}}$ which is evaluated. For $\bar{\rho} = \infty$ (untruncated ρ_t) we get the value function of the target policy V^π , whereas for finite $\bar{\rho}$, we evaluate a policy which is in between μ and π (and when ρ is close to 0, then we evaluate V^μ). So the larger $\bar{\rho}$ the smaller the bias in off-policy learning. The variance naturally grows with $\bar{\rho}$. However notice that we do not take the product of those ρ_t coefficients (in contrast to the c_s coefficients) so the variance does not explode with the time horizon.
- \bar{c} impacts the contraction modulus η of \mathcal{R} (thus the speed at which an online-algorithm like V-trace will converge to its fixed point $V^{\pi_{\bar{\rho}}}$). In terms of variance reduction, here is it really important to truncate the importance sampling ratios in c_t because we take the product of those. Fortunately, our result says that for any level of truncation \bar{c} , the fixed point (the value function $V^{\pi_{\bar{\rho}}}$ we converge to) is the same: it does not depend on \bar{c} but on $\bar{\rho}$ only.

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

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

Thus

$$\begin{aligned} \mathcal{R}V_1(x) - \mathcal{R}V_2(x) &= (1 - \mathbb{E}_\mu \rho_0) [V_1(x) - V_2(x)] + \mathbb{E}_\mu \left[\sum_{t \geq 0} \gamma^{t+1} \left(\prod_{s=0}^{t-1} c_s \right) [\rho_t - c_t \rho_{t+1}] [V_1(x_{t+1}) - V_2(x_{t+1})] \right] \\ &= \mathbb{E}_\mu \left[\sum_{t \geq 0} \gamma^t \left(\prod_{s=0}^{t-2} c_s \right) \underbrace{[\rho_{t-1} - c_{t-1} \rho_t]}_{\alpha_t} [V_1(x_t) - V_2(x_t)] \right], \end{aligned}$$

with the notation that $c_{-1} = \rho_{-1} = 1$ and $\prod_{s=0}^{t-2} c_s = 1$ for $t = 0$ and 1 . Now the coefficients $(\alpha_t)_{t \geq 0}$ are non-negative in expectation. Indeed, since $\bar{\rho} \geq \bar{c}$, we have

$$\mathbb{E}_\mu \alpha_t = \mathbb{E}[\rho_{t-1} - c_{t-1} \rho_t] \geq \mathbb{E}_\mu [c_{t-1}(1 - \rho_t)] \geq 0,$$

since $\mathbb{E}_\mu \rho_t \leq \mathbb{E}_\mu \left[\frac{\pi(a_t|x_t)}{\mu(a_t|x_t)} \right] = 1$. Thus $V_1(x) - V_2(x)$ is a linear combination of the values $V_1 - V_2$ at other states, weighted by non-negative coefficients whose sum is

$$\begin{aligned} & \sum_{t \geq 0} \gamma^t \mathbb{E}_\mu \left[\left(\prod_{s=0}^{t-2} c_s \right) [\rho_{t-1} - c_{t-1} \rho_t] \right] \\ &= \sum_{t \geq 0} \gamma^t \mathbb{E}_\mu \left[\left(\prod_{s=0}^{t-2} c_s \right) \rho_{t-1} \right] - \sum_{t \geq 0} \gamma^t \mathbb{E}_\mu \left[\left(\prod_{s=0}^{t-1} c_s \right) \rho_t \right] \\ &= \sum_{t \geq 0} \gamma^t \mathbb{E}_\mu \left[\left(\prod_{s=0}^{t-2} c_s \right) \rho_{t-1} \right] - \gamma^{-1} \left(\sum_{t \geq 0} \gamma^t \mathbb{E}_\mu \left[\left(\prod_{s=0}^{t-2} c_s \right) \rho_{t-1} \right] - 1 \right) \\ &= \underbrace{\gamma^{-1} - (\gamma^{-1} - 1) \sum_{t \geq 0} \gamma^t \mathbb{E}_\mu \left[\left(\prod_{s=0}^{t-2} c_s \right) \rho_{t-1} \right]}_{\geq 1 + \gamma \mathbb{E}_\mu \rho_0} \\ &\leq 1 - (1 - \gamma) \mathbb{E}_\mu \rho_0 \\ &\leq 1 - (1 - \gamma) \beta \\ &< 1. \end{aligned}$$

We deduce that $\|\mathcal{R}V_1(x) - \mathcal{R}V_2(x)\| \leq \eta \|V_1 - V_2\|_\infty$, with $\eta = \gamma^{-1} - (\gamma^{-1} - 1) \sum_{t \geq 0} \gamma^t \mathbb{E}_\mu \left[\left(\prod_{s=0}^{t-2} c_s \right) \rho_{t-1} \right] \leq 1 - (1 - \gamma) \beta < 1$, so \mathcal{R} is a contraction mapping. Thus \mathcal{R} possesses a unique fixed point. Let us now prove that this fixed point is $V^{\pi_{\bar{\rho}}}$. We have:

$$\begin{aligned} & \mathbb{E}_\mu [\rho_t (r_t + \gamma V^{\pi_{\bar{\rho}}}(x_{t+1}) - V^{\pi_{\bar{\rho}}}(x_t)) | x_t] \\ &= \sum_a \mu(a|x_t) \min \left(\rho, \frac{\pi(a|x_t)}{\mu(a|x_t)} \right) \left[r(x_t, a) + \gamma \sum_y p(y|x_t, a) V^{\pi_{\bar{\rho}}}(y) - V^{\pi_{\bar{\rho}}}(x_t) \right] \\ &= \underbrace{\sum_a \pi_{\bar{\rho}}(a|x_t) \left[r(x_t, a) + \gamma \sum_y p(y|x_t, a) V^{\pi_{\bar{\rho}}}(y) - V^{\pi_{\bar{\rho}}}(x_t) \right]}_{=0} \sum_b \min(\rho \mu(b|x_t), \pi(b|x_t)) \\ &= 0, \end{aligned}$$

since this is the Bellman equation for $V^{\pi_{\bar{\rho}}}$. We deduce that $\mathcal{R}V^{\pi_{\bar{\rho}}} = V^{\pi_{\bar{\rho}}}$, thus $V^{\pi_{\bar{\rho}}}$ is the unique fixed point of \mathcal{R} . \square

A.2. Online learning

Theorem 2. Assume a tabular representation, i.e. the state and action spaces are finite. Consider a set of trajectories, with the k^{th} trajectory $x_0, a_0, r_0, x_1, a_1, r_1, \dots$ generated by following μ : $a_t \sim \mu(\cdot|x_t)$. For each state x_s along this trajectory, update

$$V_{k+1}(x_s) = V_k(x_s) + \alpha_k(x_s) \sum_{t \geq s} \gamma^{t-s} (c_s \dots c_{t-1}) \rho_t (r_t + \gamma V_k(x_{t+1}) - V_k(x_t)), \quad (7)$$

with $c_i = \min(\bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)})$, $\rho_i = \min(\bar{\rho}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)})$, $\bar{\rho} \geq \bar{c}$. 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 $V_k \rightarrow V^{\pi_{\bar{\rho}}}$ almost surely.

The proof is a straightforward application of the convergence result for stochastic approximation algorithms to the fixed point of a contraction operator, see e.g. Dayan & Sejnowski (1994); Bertsekas & Tsitsiklis (1996); Kushner & Yin (2003).

A.3. On the choice of q_s in policy gradient

The policy gradient update rule (4) makes use of the coefficient $q_s = r_s + \gamma v_{s+1}$ as an estimate of $Q^{\pi_{\bar{\rho}}}(x_s, a_s)$ built from the V-trace estimate v_{s+1} at the next state x_{s+1} . The reason why we use q_s instead of v_s as target for our Q-value $Q^{\pi_{\bar{\rho}}}(x_s, a_s)$ is to make sure our estimate of the Q-value is as unbiased as possible, and the first requirement is that it is entirely unbiased in the case of perfect representation of the V-values. Indeed, assuming our value function is correctly estimated at all states, i.e. $V = V^{\pi_{\bar{\rho}}}$, then we have $\mathbb{E}[q_s|x_s, a_s] = Q^{\pi_{\bar{\rho}}}(x_s, a_s)$ (whereas we do not have this property for v_t). Indeed,

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

whereas

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

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

B. Reference Scores

Task t	Human h	Random r	Experts	IMPALA
rooms_collect_good_objects_test	10.0	0.1	9.0	5.8
rooms_exploit_deferred_effects_test	85.7	8.5	15.6	11.0
rooms_select_nonmatching_object	65.9	0.3	7.3	26.1
rooms_watermaze	54.0	4.1	26.9	31.1
rooms_keys_doors_puzzle	53.8	4.1	28.0	24.3
language_select_described_object	389.5	-0.1	324.6	593.1
language_select_located_object	280.7	1.9	189.0	301.7
language_execute_random_Task	254.1	-5.9	-49.9	66.8
language_answer_quantitative_question	184.5	-0.3	219.4	264.0
lasertag_one_opponent_large	12.7	-0.2	-0.2	0.3
lasertag_three_opponents_large	18.6	-0.2	-0.1	4.1
lasertag_one_opponent_small	18.6	-0.1	-0.1	2.5
lasertag_three_opponents_small	31.5	-0.1	19.1	11.3
natlab_fixed_large_map	36.9	2.2	34.7	12.2
natlab_varying_map_regrowth	24.4	3.0	20.7	15.9
natlab_varying_map_randomized	42.4	7.3	36.1	29.0
skymaze_irreversible_path_hard	100.0	0.1	13.6	30.0
skymaze_irreversible_path_varied	100.0	14.4	45.1	53.6
pyschlab_arbitrary_visuomotor_mapping	58.8	0.2	16.4	14.3
pyschlab_continuous_recognition	58.3	0.2	29.9	29.9
pyschlab_sequential_comparison	39.5	0.1	0.0	0.0
pyschlab_visual_search	78.5	0.1	0.0	0.0
explore_object_locations_small	74.5	3.6	57.8	62.6
explore_object_locations_large	65.7	4.7	37.0	51.1
explore_obstructed_goals_small	206.0	6.8	135.2	188.8
explore_obstructed_goals_large	119.5	2.6	39.5	71.0
explore_goal_locations_small	267.5	7.7	209.4	252.5
explore_goal_locations_large	194.5	3.1	83.1	125.3
explore_object_rewards_few	77.7	2.1	39.8	43.2
explore_object_rewards_many	106.7	2.4	58.7	62.6
Mean Capped Normalised Score: $(\sum_t \min[1, (s_t - r_t)/(h_t - r_t)]) / N$	100%	0%	44.5%	49.4%

Table B.1. DMLab-30 test scores.

B.1. Final training scores on DMLab-30

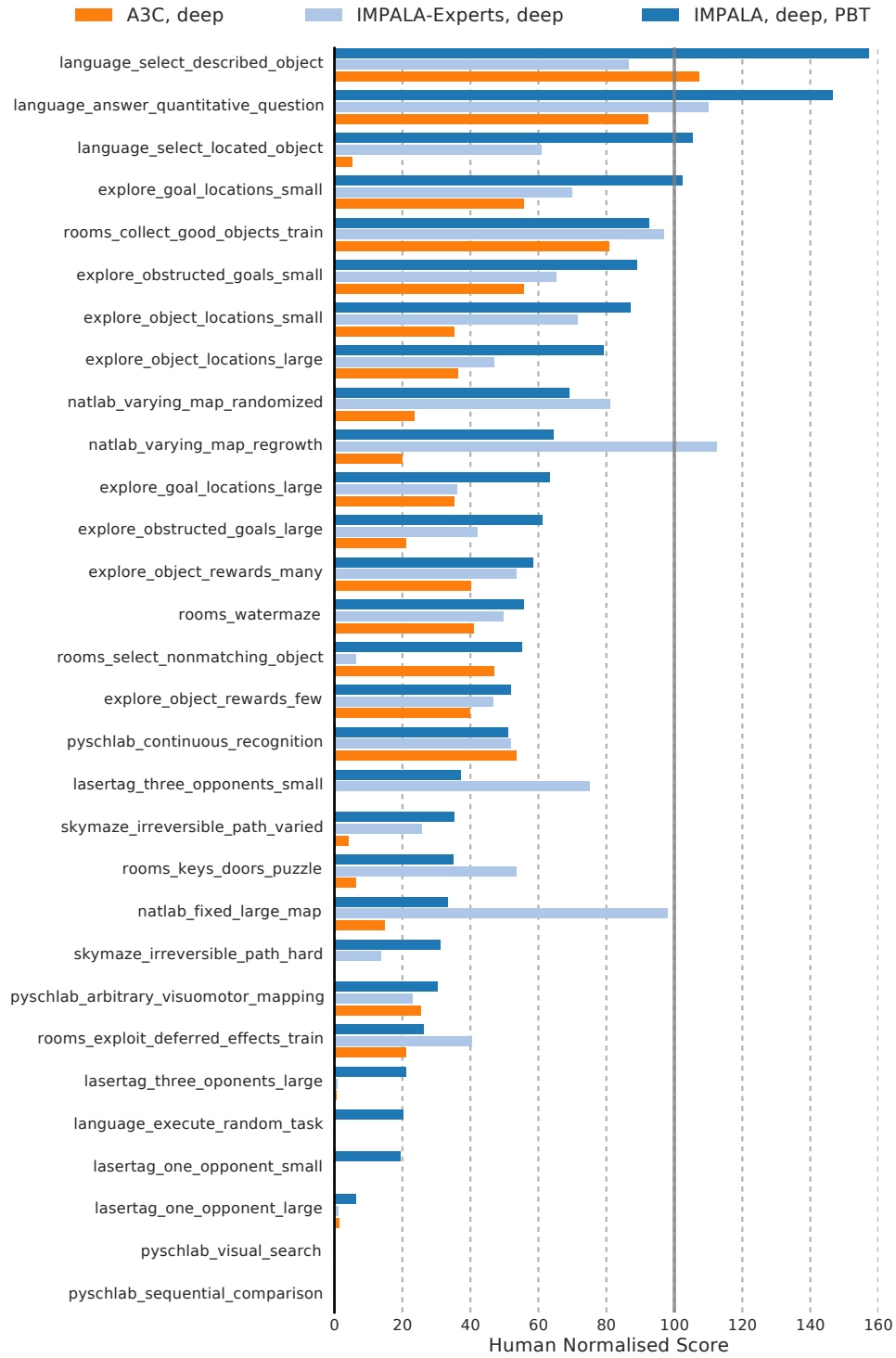


Figure B.1. Human normalised scores across all DMLab-30 tasks.

C. Parameters

In this section, the specific parameter settings that are used throughout our experiments are given in detail.

Hyperparameter	Range	Distribution
Entropy regularisation	$[5e-5, 1e-2]$	Log uniform
Learning rate	$[5e-6, 5e-3]$	Log uniform
RMSProp epsilon (ϵ) regularisation parameter	$[1e-1, 1e-3, 1e-5, 1e-7]$	Categorical

Table C.1. The ranges used in sampling hyperparameters across all experiments that used a sweep and for the initial hyperparameters for PBT. Sweep size and population size are 24. Note, the loss is *summed* across the batch and time dimensions.

Action	Native DeepMind Lab Action
Forward	[0, 0, 0, 1, 0, 0, 0, 0]
Backward	[0, 0, 0, -1, 0, 0, 0, 0]
Strafe Left	[0, 0, -1, 0, 0, 0, 0, 0]
Strafe Right	[0, 0, 1, 0, 0, 0, 0, 0]
Look Left	[-20, 0, 0, 0, 0, 0, 0, 0]
Look Right	[20, 0, 0, 0, 0, 0, 0, 0]
Forward + Look Left	[-20, 0, 0, 1, 0, 0, 0, 0]
Forward + Look Right	[20, 0, 0, 1, 0, 0, 0, 0]
Fire	[0, 0, 0, 0, 0, 1, 0, 0]

Table C.2. Action set used in all tasks from the DeepMind Lab environment, including the DMLab-30 experiments.

C.1. Fixed Model Hyperparameters

In this section, we list all the hyperparameters that were kept fixed across all experiments in the paper which are mostly concerned with observations specifications and optimisation. We first show below the reward pre-processing function that is used across all experiments using DeepMind Lab, followed by all fixed numerical values.

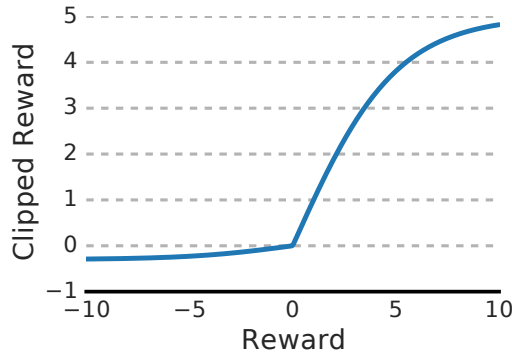


Figure C.1. Optimistic Asymmetric Clipping - $0.3 \cdot \min(\tanh(\text{reward}), 0) + 5.0 \cdot \max(\tanh(\text{reward}), 0)$

Parameter	Value
Image Width	96
Image Height	72
Action Repetitions	4
Unroll Length	100
Reward Clipping	
- Single Levels	[-1, 1]
- DMLab-30, including experts	See Figure C.1
Discount (γ)	0.99
Baseline loss scaling	0.5
RMSPProp momentum	0.0
Experience Replay (in Section 5.2.2)	
- Capacity	10,000 trajectories
- Sampling	Uniform
- Removal	First-in-first-out

Table C.3. Fixed model hyperparameters across all DeepMind Lab experiments.

D. V-trace Analysis

D.1. Controlled Updates

Here we show how different algorithms (On-Policy, No-correction, ϵ -correction, V-trace) behave under varying levels of policy-lag between the actors and the learner.



Figure D.1. As the policy-lag (the number of update steps the actor policy is behind learner policy) increases, learning with V-trace is more robust compared to ϵ -correction and pure on-policy learning.

D.2. V-trace Stability Analysis

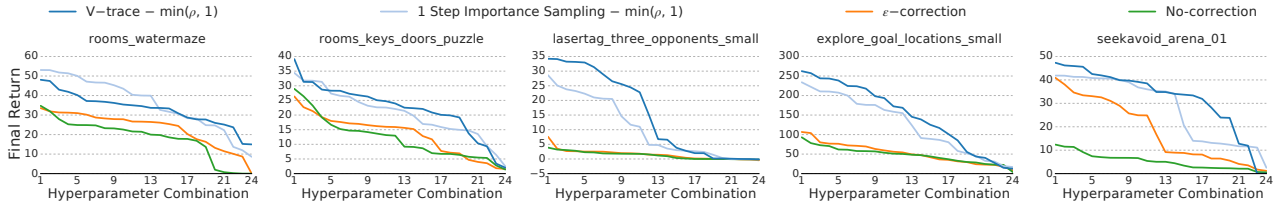


Figure D.2. Stability across hyper parameter combinations for different off-policy correction variants using replay. V-trace is much more stable across a wide range of parameter combinations compared to ϵ -correction and pure on-policy learning.

E. Population Based Training

For Population Based Training we used a “burn-in” period of 20 million frames where no evolution is done. This is to stabilise the process and to avoid very rapid initial adaptation which hinders diversity. After collecting 5,000 episode rewards in total, the mean capped human normalised score is calculated and a random instance in the population is selected. If the score of the selected instance is more than an absolute 5% higher, then the selected instance weights and parameters are adopted.

No matter if an adaptation happened or not, each parameter (RMSProp epsilon, learning rate and entropy cost) is permuted with 33% probability by multiplying 1.2 or 1/1.2.

This is different from Jaderberg et al. (2017) in that our multiplication is unbiased where they use a multiplication of 1.2 or .8. Furthermore, they only permute parameters in case an adaptation happens. We found that diversity is increased when the parameters are permuted even if no adaptation happened.

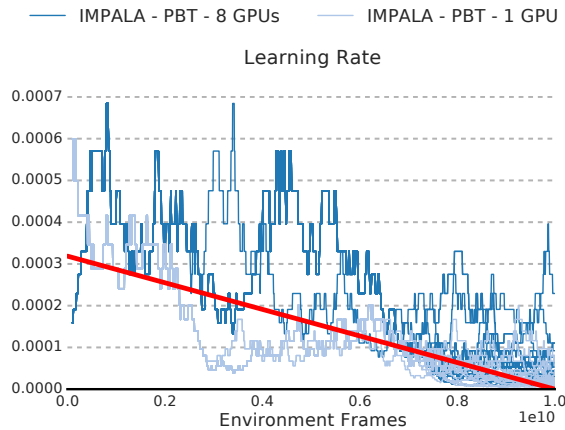


Figure E.1. Learning rate schedule that is discovered by the PBT Jaderberg et al. (2017) method.

F. Atari Experiments

All agents trained on Atari are equipped only with a feed forward network and pre-process frames in the same way as described in Mnih et al. (2016). When training experts agents, we use the same hyperparameters for each level for both IMPALA and A3C. These hyperparameters are the result of tuning A3C with a shallow network on the following games: breakout, pong, space_invaders, seaquest, beam_rider, qbert. Following related work, experts use game-specific action sets.

The multi-task agent was equipped with a feed forward residual network (see Figure 4). Its learning rate, entropy regularisation, RMSProp ϵ and gradient clipping threshold were adapted through population based training. To be able to use the same policy layer on all Atari games in the multi-task setting we train the multi-task agent on the full Atari action set consisting of 18 actions.

Agents were trained using the following set of hyperparameters:

Parameter	Value
Image Width	84
Image Height	84
Grayscale	Yes
Action Repetitions	4
Max-pool over last N action repeat frames	2
Frame Stacking	4
End of episode when life lost	Yes
Reward Clipping	[-1, 1]
Unroll Length	20
Batch size	32
Discount (γ)	0.99
Baseline loss scaling	0.5
Entropy Regularizer	0.01
RMSProp momentum	0.0
RMSProp ϵ	0.01
Learning rate	0.0006
Clip global gradient norm	40.0
Learning rate schedule	Anneal linearly to 0 From beginning to end of training.
Population based training (only multi-task agent)	
- Population size	24
- Start parameters	Same as DMLab-30 sweep
- Fitness	Mean capped human normalised scores ($\sum_l \min [1, (s_t - r_t) / (h_t - r_t)]$) / N
- Adapted parameters	Gradient clipping threshold Entropy regularisation Learning rate RMSProp ϵ

Table F.1. Hyperparameters for Atari experiments.

References

- Bertsekas, Dimitri P. and Tsitsiklis, John N. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- Dayan, Peter and Sejnowski, Terrence J. TD(λ) converges with probability 1. *Machine Learning*, 14(1):295–301, 1994. doi: 10.1023/A:1022657612745.
- Jaderberg, Max, Dalibard, Valentin, Osindero, Simon, Czarnecki, Wojciech M., Donahue, Jeff, Razavi, Ali, Vinyals, Oriol, Green, Tim, Dunning, Iain, Simonyan, Karen, Fernando, Chiranth, and Kavukcuoglu, Koray. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.
- Kushner, H. and Yin, G.G. *Stochastic Approximation and Recursive Algorithms and Applications*. Stochastic Modelling and Applied Probability. Springer New York, 2003. ISBN 9780387008943.
- Mnih, Volodymyr, Badia, Adrià Puigdomènech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P., Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016.