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

**Lasse Espeholt** [*1] **Hubert Soyer** [*1] **Remi Munos** [*1] **Karen Simonyan** [1] **Volodymyr Mnih** [1] **Tom Ward** [1] **Yotam Doron** [1] **Vlad Firoiu** [1] **Tim Harley** [1] **Iain Dunning** [1] **Shane Legg** [1] **Koray Kavukcuoglu** [1]

## 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. We have developed a new distributed agent IMPALA (Importance Weighted Actor-Learner Architecture) that not only uses resources more efficiently in single-machine training but also scales to thousands of machines without sacrificing data efficiency or resource utilisation. We achieve stable learning at high throughput by combining decoupled acting and learning with a novel off-policy correction method called V-trace. 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 with 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 **Imp**ortance Weighted **A**ctor-**L**earner **A**rchitecture (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 updates on mini-batches of trajectories while aggressively parallelising all time independent 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 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 30 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 by training a single agent 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 and by training a single agent on all games in the Atari-57 set of tasks.

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

*Proceedings of the $35^{th}$ International Conference on Machine Learning*, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

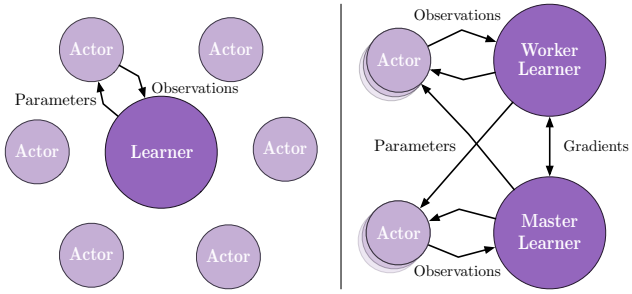


*Figure 1.* **Left: Single Learner.** Each *actor* generates trajectories and sends them via a queue to the *learner*. Before starting the next trajectory, *actor* retrieves the latest policy parameters from *learner*. **Right: Multiple Synchronous Learners.** Policy parameters are distributed across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

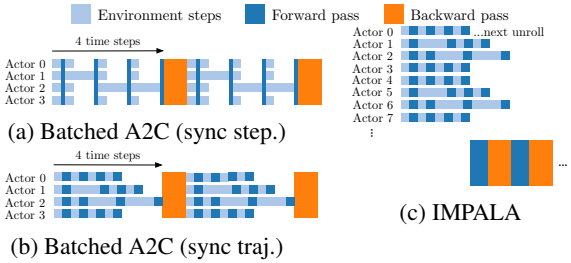


*Figure 2.* Timeline for one unroll with 4 steps using different architectures. Strategies shown in **(a)** and **(b)** can lead to low GPU utilisation due to rendering time variance within a batch. In **(a)**, the actors are synchronised after every step. In **(b)** after every $n$ steps. IMPALA **(c)**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

## 2. Related Work

The earliest attempts to scale up deep reinforcement learning relied on distributed asynchronous SGD (Dean et al., 2012) with multiple workers. Examples include distributed A3C (Mnih et al., 2016) and Gorila (Nair et al., 2015), a distributed version of Deep Q-Networks (Mnih et al., 2015). Recent alternatives to asynchronous SGD for RL include using evolutionary processes (Salimans et al., 2017), distributed BA3C (Adamski et al., 2018) and Ape-X (Horgan et al., 2018) which has a distributed replay but a synchronous learner.

There have also been multiple efforts that scale up reinforcement learning 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 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 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. The 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; O'Donoghue et al., 2017) 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 (Wang et al., 2017; 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 A3C learn a state-value function $V$ instead of a state-action-value function $Q$. V-trace is based on the state-value function.

## 3. IMPALA

IMPALA (Figure 1) uses an actor-critic setup to learn a policy $\pi$ and a baseline function $V^\pi$. The process of generating experiences is decoupled from learning the parameters of $\pi$ and $V^\pi$. 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 $\pi$ off-policy.

At the beginning of each trajectory, an actor updates its own local policy $\mu$ to the latest learner policy $\pi$ and runs it for $n$ steps in its environment. After $n$ steps, the actor sends the trajectory of states, actions and rewards $x_1, a_1, r_1, \ldots, x_n, a_n, r_n$ 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 $\pi$ on batches of trajectories, each collected from many actors. This simple architecture enables the learner(s) to be accelerated using GPUs and actors to be easily distributed across many machines. However, the learner policy $\pi$ is potentially several updates ahead of the actor's policy $\mu$ at the time of update, therefore there is a *policy-lag* between the actors and learner(s). V-trace cor-

rects for this lag to achieve extremely high data throughput while maintaining data efficiency. Using an actor-learner architecture, provides fault tolerance like distributed A3C but often has lower communication overhead since the actors send observations rather than parameters/gradients.

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 distributed set of learners to train large neural networks efficiently as shown in Figure 1. Parameters are distributed across the learners and actors retrieve the parameters from all the learners in parallel while only sending observations to a single learner. IMPALA use synchronised parameter update which is vital to maintain data efficiency when scaling to many 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. As an example, a typical deep RL agent features a convolutional network followed by a 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. Similarly, it also applies the output layer to all time steps in parallel once all LSTM states are computed. This optimisation increases the effective batch size to thousands. LSTM-based agents also obtain significant speedups on the learner by exploiting the network structure dependencies and operation fusion (Appleyard et al., 2016).

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 the 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 $\pi$ that maximises the expected sum of future discounted rewards: $V^\pi(x) \stackrel{\text{def}}{=} \mathbb{E}_\pi\big[\sum_{t\geq 0}\gamma^t r_t\big]$, 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 $a_t \sim \pi(\cdot|x_t)$ is the action generated by following some policy $\pi$.

The goal of an off-policy RL algorithm is to use trajectories generated by some policy $\mu$, called the *behaviour policy*, to learn the value function $V^\pi$ of another policy $\pi$ (possibly different from $\mu$), 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 $\mu$. We define the $n$-steps V-trace target for $V(x_s)$, our value approximation at state $x_s$, as:

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

where $\delta_t V \stackrel{\text{def}}{=} \rho_t\big(r_t + \gamma V(x_{t+1}) - V(x_t)\big)$ is a temporal difference for $V$, and $\rho_t \stackrel{\text{def}}{=} \min\big(\bar\rho, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}\big)$ and $c_i \stackrel{\text{def}}{=} \min\big(\bar c, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}\big)$ 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

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

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 $\rho_t$ play different roles. The weight $\rho_t$ 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 $\mu$), is the value function $V^{\pi_{\bar\rho}}$ of some policy $\pi_{\bar\rho}$, defined by

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

(see the analysis in Appendix A). So when $\bar\rho$ is infinite (i.e. no truncation of $\rho_t$), then this is the value function $V^\pi$ 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 $\mu$ and $\pi$. At the limit when $\bar{\rho}$ is close to zero, we obtain the value function of the behaviour policy $V^{\mu}$. 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 $\pi$ and $\mu$ 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:*

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

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

### 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 $\mu$ is

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

where $Q^{\mu}(x_s, a_s) \stackrel{\text{def}}{=} \mathbb{E}_{\mu}\big[\sum_{t\geq s}\gamma^{t-s} r_t | x_s, a_s\big]$ is the state-action value of policy $\mu$ at $(x_s, a_s)$. This is usually implemented by a stochastic gradient ascent that updates the policy parameters in the direction of $\mathbb{E}_{a_s\sim\mu(\cdot|x_s)}\Big[\nabla \log \mu(a_s|x_s) q_s \big| x_s\Big]$, 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 $\mu$.

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 $\mu$, to update our policy parameter in the direction of

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

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 for analysis and Appendix E.3 for a comparison of different ways to estimate $q_s$.

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

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

and the policy parameters $\omega$ in the direction of the policy gradient:

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

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 train agents—each with one set of weights for all tasks—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)

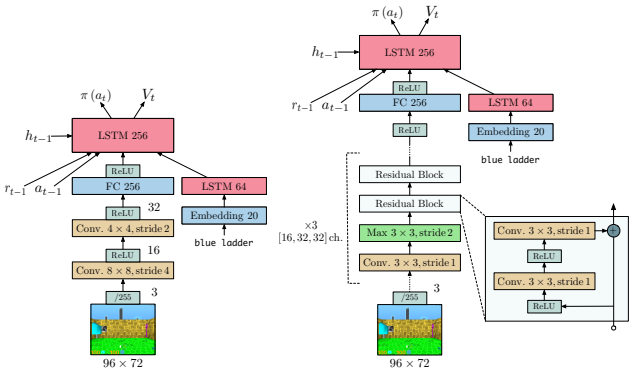


*Figure 3.* Model Architectures. **Left:** Small architecture, 2 convolutional layers and 1.2 million parameters. **Right:** Large architecture, 15 convolutional layers and 1.6 million parameters.

| **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.

*Table 1.* Throughput on `seekavoid_arena_01` (task 1) and `rooms_keys_doors_puzzle` (task 2) with the shallow model in Figure 3. The latter has variable length episodes and slow restarts. Batched A2C and IMPALA use batch size 32 if not otherwise mentioned.

with an LSTM before the policy and value (shown in Figure 3 (left)) and a deeper residual model (He et al., 2016) (shown in Figure 3 (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. For single-machine experiments using GPUs, we use dynamic batching in the forward pass to avoid several batch size 1 forward passes. Our dynamic batching module is implemented by specialised TensorFlow operations but is conceptual similar to the queues used in GA3C. Table 1 details the results for single-machine and multi-machine versions with the shallow model from Figure 3. In the single-machine case, IMPALA achieves the highest 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. With the optimisations from Section 3.1 to speed up the GPU-based learner, the IMPALA agent achieves a throughput rate of 250,000 frames/sec or 21 billion frames/day. Note, 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 D.1. The other hyperparameters were fixed to values specified in Appendix D.3.

#### 5.2.1. Convergence and Stability

Figure 4 shows a comparison between IMPALA, A3C and batched A2C with the shallow model in Figure 3. 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.

In addition to reaching better final performance, IMPALA is also more robust to the choice of hyperparameters than A3C. Figure 4 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.

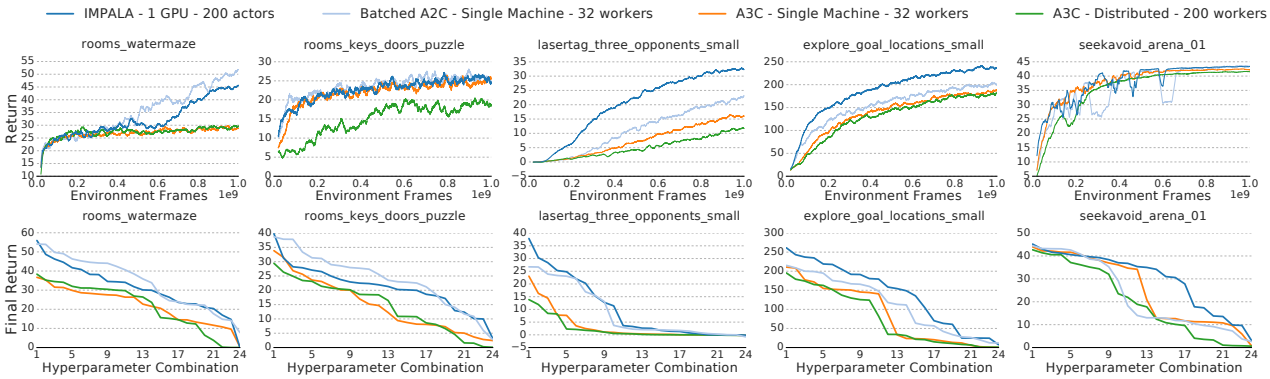


*Figure 4.* **Top Row:** Single task training on 5 DeepMind Lab tasks. Each curve is the mean of the best 3 runs based on final return. IMPALA achieves better performance than A3C. **Bottom Row:** Stability across hyperparameter combinations sorted by the final performance across different hyperparameter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

| | 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 |
| $\varepsilon$-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 |
| $\varepsilon$-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 $\varepsilon$-correction/No-correction. However, when the lag increases due to use of experience replay, V-trace performs better than all other methods in 4 out 5 tasks.

**2. $\varepsilon$-correction** - Add a small value ($\varepsilon = 1e\text{-}6$) during 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 $\rho_t$ and $c_t$ at 1 (i.e. $\bar{c} = \bar{\rho} = 1$). This 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 $\pi$ and $\mu$. 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, $\varepsilon$-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 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. Also note that V-trace is the only variant that consistently benefits from adding experience replay. $\varepsilon$-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 E.1 shows results of a more detailed analysis. Figure E.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. Note, the model does not know which task it is being trained or evaluated on.

| 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.

#### 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 on DeepMind Lab. Among the many task types in the suite are visually complex environments with natural-looking terrain, instruction-based tasks with grounded language (Hermann et al., 2017), navigation tasks, cognitive (Leibo et al., 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 D.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 $\varepsilon$. See Appendix F 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 3 (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 (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 for each task separately but instead across all tasks on which the 30 expert agents were trained.

Table 3 and Figure 5 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. 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. 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 5 shows, the multi-task version is outperforms *IMPALA-Experts* throughout training and the breakdown into individual scores in Appendix B shows positive transfer on tasks such as language tasks and laser tag tasks.

Comparing A3C to IMPALA with respect to wall clock time (Figure 6) 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 of the deep model by a factor of 7 to 210K frames/sec, up from 30K frames/sec.

#### 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 convolutional layers. 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 G.

In addition to individual per-game experts, trained for 200 million frames with a fixed set of hyperparameters, 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. For the Atari-57 agent, we use population based training with a population size of 24 to adapt entropy regularisation, learning rate, RMSProp $\varepsilon$ and the global gradient norm clipping threshold throughout training.

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

*Figure 5.* Performance of best agent in each sweep/population during training on the DMLab-30 task-set wrt. data consumed across all environments. 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.

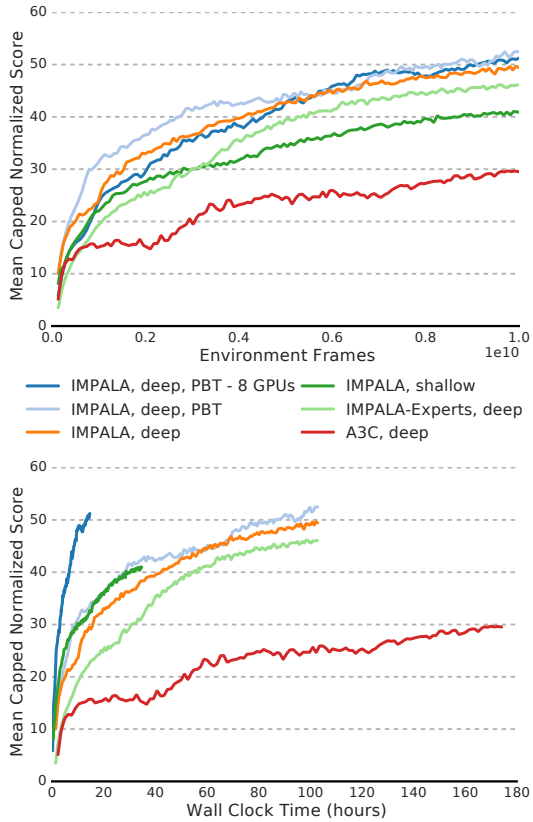


*Figure 6.* Performance on DMLab-30 wrt. wall-clock time. All models used the deep architecture (Figure 3). The high throughput of IMPALA results in orders of magnitude faster learning.

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 million 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

| Human Normalised Return | Median | Mean |
|---|---|---|
| A3C, shallow, experts | 54.9% | 285.9% |
| A3C, deep, experts | 117.9% | 503.6% |
| The Reactor | 187% | N/A |
| 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. For a level-by-level comparison to ACKTR (Wu et al., 2017) and Reactor see Appendix C.1.

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. With its simple but scalable distributed architecture, IMPALA can make efficient use of available compute at small and large scale. This 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. 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 on DMLab-30). 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 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, Jacob Menick and David Silver for their comments and insightful discussions.

## References

Abadi, M., Isard, M., and Murray, D. 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.

Adamski, I., Adamski, R., Grel, T., Jedrych, A., Kaczmarek, K., and Michalewski, H. Distributed deep reinforcement learning: Learn how to play atari games in 21 minutes. *CoRR*, abs/1801.02852, 2018.

Appleyard, J., Kociský, T., and Blunsom, P. Optimizing performance of recurrent neural networks on gpus. *CoRR*, abs/1604.01946, 2016.

Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., and Kautz, J. GA3C: GPU-based A3C for deep reinforcement learning. *NIPS Workshop*, 2016.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Tirumala, D., Muldal, A., Heess, N., and Lillicrap, T. Distributional policy gradients. *ICLR*, 2018.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Kuttler, H., Lefrancq, A., Green, S., Valdes, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., and Petersen, S. Deepmind lab. *CoRR*, abs/1612.03801, 2016.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013a.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013b.

Chen, J., Monga, R., Bengio, S., and Józefowicz, R. Revisiting distributed synchronous SGD. *CoRR*, abs/1604.00981, 2016.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

Clemente, A. V., Martínez, H. N. C., and Chandra, A. Efficient parallel methods for deep reinforcement learning. *CoRR*, abs/1705.04862, 2017.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., and Ng, A. Y. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*, pp. 1223–1231, 2012.

Geist, M. and Scherrer, B. Off-policy learning with eligibility traces: A survey. *The Journal of Machine Learning Research*, 15(1):289–333, 2014.

Gruslys, A., Dabney, W., Azar, M. G., Piot, B., Bellemare, M. G., and Munos, R. The Reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. *ICLR*, 2018.

Harutyunyan, A., Bellemare, M. G., Stepleton, T., and Munos, R. *Q(λ) with Off-Policy Corrections*, pp. 305–320. Springer International Publishing, Cham, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W., Jaderberg, M., Teplyashin, D., et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., and Silver, D. Distributed prioritized experience replay. *ICLR*, 2018.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017a.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *ICLR*, 2017b.

Leibo, J. Z., d'Autume, C. d. M., Zoran, D., Amos, D., Beattie, C., Anderson, K., Castañeda, A. G., Sanchez, M., Green, S., Gruslys, A., et al. Psychlab: A psychology laboratory for deep reinforcement learning agents. *arXiv preprint arXiv:1801.08116*, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. *ICML*, 2016.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1046–1054, 2016.

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., Maria, A. D., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., and Silver, D. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015.

O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. Combining policy gradient and Q-learning. In *ICLR*, 2017.

Precup, D., Sutton, R. S., and Singh, S. Eligibility traces for off-policy policy evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

Precup, D., Sutton, R. S., and Dasgupta, S. Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Laerning*, pp. 417–424, 2001.

Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Salimans, T., Ho, J., Chen, X., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G. v. d., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 10 2017. ISSN 0028-0836. doi: 10.1038/nature24270.

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

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. Sample efficient actor-critic with experience replay. In *ICLR*, 2017.

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

Wu, Y., Mansimov, E., Liao, S., Grosse, R. B., and Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *CoRR*, abs/1708.05144, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.