The Reactor: A Sample-Efficient Actor-Critic Architecture

Audrunas Gruslys [1] **Mohammad Gheshlaghi Azar** [1] **Marc G. Bellemare** [1] **Remi Munos** [1]

Abstract

In this work we present a new reinforcement learning agent, called *Reactor* (for Retrace-actor), based on an off-policy multi-step return actor-critic architecture. The agent uses a deep recurrent neural network for function approximation. The network outputs a target policy π (the actor), an action-value Q-function (the critic) evaluating the current policy π, and an estimated behavioural policy $\hat{\mu}$ which we use for off-policy correction. The agent maintains a memory buffer filled with past experiences. The critic is trained by the multi-step off-policy *Retrace algorithm* and the actor is trained by a novel β-*leave-one-out policy gradient* estimate (which uses both the off-policy corrected return and the estimated Q-function). The Reactor is sample-efficient thanks to the use of memory replay, and numerical efficient since it uses multi-step returns. Also both acting and learning can be parallelized. We evaluated our algorithm on 57 Atari 2600 games and demonstrate that it achieves state-of-the-art performance.

1. Introduction

Model free deep reinforcement learning has achieved remarkable success lately in many domains ranging from achieving human and super-human level control in video games (Mnih et al., 2016), (Mnih et al., 2015) to continuous motor control tasks (Lillicrap et al., 2015). However, two of the most popular frameworks, DQN (Mnih et al., 2015) and A3C (Mnih et al., 2016), have several disadvantages: DQN suffers from potentially slow learning caused by single-step temporal difference updates, while A3C has a relatively low data-efficiency (since it does not use a memory buffer) and requires being trained on several copies of the same environment simultaneously. Also A3C is an on-policy algorithm which does not separate the evaluation policy from the behaviour policy, making it difficult to explore safely. Data-efficiency and off-policy learning are arguably important for many real-world problems where interactions with the environment is expensive. For this reason we wanted to exploit the off-policy advantage of DQN while also getting the benefits of using multi-step returns and an explicit policy evaluation of A3C framework.

In this work we introduce a new agent architecture, called *Reactor* (for Retrace-actor), which has the following features. The Reactor uses an off-policy multi-step learning algorithm based on an actor-critic architecture. The critic implements the Retrace (Munos et al., 2016) algorithm, while the actor is trained by a new policy gradient algorithm, called β-*leave-one-out*, which makes use of both the off-policy Retrace-corrected return and the estimated Q-function.

The Reactor is sample-efficient thanks to the use of memory replay, and numericaly efficient since it uses multi-step returns which improves the speed of reward propagation backwards in time, while the off-policy nature of the algorithm enables to reuse old data and be sample efficient.

The reason we use an actor-critic architecture is to explicitly evaluate behavioural policies which may differ from policies derived from learned Q-values (such as epsilon-greedy policies). This can be beneficial for several reasons. Firstly, a policy may be easier to learn than action values or action advantages. A second advantage is that estimating an explicit behavioural policy may reduce the problem of Q-value over-estimation when bootstrapping in the presence of noise, as the best action and the actual action value are evaluated by different functional approximations. Similar reasoning led to the introduction of Double DQN algorithm (Van Hasselt et al., 2016). The third advantage is that evaluating non-deterministic policies can potentially lead to less trace cutting while using Retrace algorithm. For this reason it is beneficial to be stochastic when actions are of equal value and the easiest way to do this is to have an explicit actor which is trained with some entropy reward.

A main difference compared to DQN is that in order to implement the multi-step Retrace algorithm, we need to replay sequences of (instead of individual) transitions. Since we do so, using a recurrent network architecture comes at no cost. Actually, using a recurrent network architecture avoids complications related to frame-stacking and gives computational gains, as each frame has to be processed by

[1]DeepMind, London, UK.

a convolutional network only once in each sequence. While using a recurrent network architecture may not be required in Atari, it may become useful in more complex partially observable domains. And even in Atari domain, recent work (Harb & Precup) showed that using LSTM (combined with Watkins Q(λ) algorithm) can improve learning compared to feed-forward networks.

Lastly, we decoupled acting from learning by allowing the actor and the learner to run in parallel. This does not block the actor during batched learning and does not block the learner while the actor is acting which in turn allows to achieve greater CPU utilization.

Related works: Like A3C (Mnih et al., 2016), Reactor is an actor-critic multi-step returns algorithm. Reactor differs from A3C in that it uses a memory buffer and an off-policy learning algorithm. Compared to DQN, Reactor brings the multi-step returns and the actor-critic architecture. DDPG (Lillicrap et al., 2015) uses an actor-critic architecture but in a continuous control setting making use of the deterministic policy gradient algorithm and does not consider multi-step returns. UNREAL agent (Jaderberg et al., 2016) improved final performance and data efficiency of A3C framework by introducing replay memory and unsupervised auxiliary tasks.

The closest work to ours is the ACER algorithm (Wang et al., 2017) which is also an actor-critic based on the Retrace algorithm which makes use of memory replay. The main differences are (1) Reactor use a different policy gradient algorithm for the actor (see discussion in Section 2.2), (2) ACER mixes on-policy (from current run) and off-policy (from memory) updates whereas Reactor uses off-policy updates replayed from the memory exclusively, (3) Reactor makes use of an estimated $\hat{\mu}$ behaviour policy instead of storing μ in memory, and (4) Reactor uses an actor and learner that can run in parallel independently from each other.

2. The actor-critic algorithm

In this section we first describe the general algorithm we used for the critic (Retrace) and the actor (β-leave-one-out). We describe our specific implementation in the next section. We first define some notation. We consider a Markov decision process with state space X and finite action space A. A (stochastic) policy is a mapping from X to distributions over actions. We consider a γ-discounted infinite-horizon criterion and define for any policy π, the Q-value of any state-action pair (x, a) as

$$Q^\pi(x,a) \stackrel{\text{def}}{=} \mathbb{E}\Big[\sum_{t\geq 0}\gamma^t r(x_t,a_t)|x_0=x, a_0=a, \pi\Big],$$

where $(\{x_t\}_{t\geq 0})$ is a trajectory generated by choosing a in x and following π thereafter, i.e., $a_t \sim \pi(\cdot|x_t)$ (for $t \geq 1$), and $r(x_t, a_t)$ is a reward function. Our goal is to find an optimal policy π^*, which maximizes $Q^\pi(x, a)$. $Q^*(x, a) = \max_\pi Q^\pi(x, a)$ represents the optimal Q-values. We have the property that any policy defined in any state x as $\in \arg\max_a Q^*(x, a)$ is an optimal policy.

2.1. The critic: Retrace(λ)

For the critic we use the Retrace(λ) algorithm introduced in (Munos et al., 2016). This is a general off-policy RL algorithm which uses multi-step returns. Assume that some trajectory $\{x_0, a_0, r_0, x_1, a_1, r_1, \ldots, x_t, a_t, r_t, \ldots,\}$ has been generated according to some *behaviour policy* μ, i.e., $a_t \sim \mu(\cdot|x_t)$. Now we want to evaluate the value of a different *target policy* π, i.e. we want to estimate Q^π. The Retrace algorithm will update our current estimate Q of Q^π in the direction of

$$\Delta Q(x_t,a_t) \stackrel{\text{def}}{=} \sum_{s\geq t}\gamma^{s-t}(c_{t+1}\ldots c_s)\delta_s^\pi Q, \qquad (1)$$

where $\delta_s^\pi Q \stackrel{\text{def}}{=} r_s + \gamma\mathbb{E}_\pi[Q(x_{s+1},\cdot)] - Q(x_s,a_s)$ is the temporal difference of Q at time s under policy π, and

$$c_s = \lambda \min\big(1, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}\big). \qquad (2)$$

The Retrace algorithm comes with the theoretical guarantee that in finite state and action spaces, repeatedly updating our current estimate Q according to (1) produces a sequence which converges to Q^π for a fixed π or to Q^* if we consider a sequence of policies π which become increasing greedy w.r.t. the Q estimates, see (Munos et al., 2016).

2.2. The actor: β-LOO policy gradient algorithm

The Reactor architecture represents both a policy $\pi(a|x)$ and Q-values $Q(x, a)$. We use a policy gradient algorithm to train the actor π which makes use of our current estimate $Q(x, a)$ of $Q^\pi(x, a)$. Let $V^\pi(x_0)$ be the value function at some initial state x_0, the policy gradient theorem (Sutton et al., 2000) says that $\nabla V^\pi(x_0) = \mathbb{E}\big[\sum_t \gamma^t \sum_a Q^\pi(x_t, a)\nabla\pi(a|x_t)\big]$, where ∇ refers to the gradient w.r.t. policy parameters. We now consider several possible ways to estimate this gradient.

To simplify notation, we drop the dependence on the state x for now and consider the problem of estimating the quantity

$$G = \sum_a Q^\pi(a)\nabla\pi(a). \qquad (3)$$

Since we consider the off-policy case, consider estimating G using a single action A drawn from a (possibly different from π) behaviour distribution $A \sim \mu$. Let us assume that

for the chosen action A we have access to an estimate $R(A)$ of $Q^\pi(A)$. Then we can use likelihood ratio (LR) method combined with an importance sampling (IS) ratio (which we call ISLR) to build an unbiased estimate of G:

$$\hat{G}_{\text{ISLR}} = \frac{\pi(A)}{\mu(A)}(R(A) - V)\nabla \log \pi(A),$$

where V is a baseline that depend on the state but not on the chosen action. However this estimate suffers from high variance. A possible way for reducing variance is to estimate G directly from (3) by using the return $R(A)$ for the chosen action A and our current estimate Q of Q^π for the other actions, which lead to the so-called *leave-one-out* (LOO) policy gradient estimate:

$$\hat{G}_{\text{LOO}} = R(A)\nabla\pi(A) + \sum_{a \neq A} Q(a)\nabla\pi(a). \tag{4}$$

This estimate has low variance but may be biased if the estimated Q values differ from Q^π. A better bias-variance tradeoff may be obtained by considering the more general *β-LOO policy gradient* estimate:

$$\hat{G}_{\beta\text{-LOO}} = \beta(R(A) - Q(A))\nabla\pi(A) + \sum_a Q(a)\nabla\pi(a), \tag{5}$$

where $\beta = \beta(\mu, \pi, A)$ can be a function of both policies π and μ and the selected action A. Notice that when $\beta = 1$, (5) reduces to (4), and when $\beta = 1/\mu(A)$, then (5) is

$$\hat{G}_{\frac{1}{\mu}\text{-LOO}} = \frac{\pi(A)}{\mu(A)}(R(A) - Q(A))\nabla \log \pi(A) + \sum_a Q(a)\nabla\pi(a). \tag{6}$$

This estimate is unbiased and can be seen as a generalization of $\hat{G}_{\text{ISLR}}$ where instead of using a state-only dependent baseline, we use a state-and-action-dependent baseline (our current estimate Q) and add the correction term $\sum_a \nabla\pi(a)Q(a)$ to cancel the bias. We now analyze the bias of the $G_{\beta\text{-LOO}}$ estimate.

Proposition 1. *Assume $A \sim \mu$ and that $\mathbb{E}[R(A)] = Q^\pi(A)$. Then, the bias of $G_{\beta\text{-LOO}}$ is $\left|\sum_a (1 - \mu(a)\beta(a))\nabla\pi(a)[Q(a) - Q^\pi(a)]\right|$.*

Proof. We have that $\mathbb{E}[\hat{G}]_{\beta\text{-LOO}}$ is

$$= \sum_a \mu(a)[\beta(a)(\underbrace{\mathbb{E}[R(a)]}_{=Q^\pi(a)} - Q(a))]\nabla\pi(a) + \sum_a Q(a)\nabla\pi(a)$$

$$= G + \sum_a (1 - \mu(a)\beta(a))[Q(a) - Q^\pi(a)]\nabla\pi(a)$$

□

Thus the bias is small when $\beta(a)$ is close to $1/\mu(a)$ (and unbiased for $\hat{G}_{\frac{1}{\mu}\text{-LOO}}$ whatever the Q estimates are) or when the Q-estimates are close to the true Q^π values. Now its variance is low when β is small. So in order to have a good bias-variance tradeoff we recommend using the β-LOO estimate with β defined as: $\beta(A) = \min\left(c, \frac{1}{\mu(A)}\right)$, for some constant $c \geq 1$. Note that $\beta(A) = 1$ when $c = 1$.

This truncated $1/\mu$ coefficient shares similarities with the truncated IS gradient estimate introduced in (Wang et al., 2017) (which we call TISLR for truncated-ISLR):

$$\hat{G}_{\text{TISLR}} = \min\left(c, \frac{\pi(A)}{\mu(A)}\right)(R(A) - V)\nabla \log \pi(A) + \sum_a \left(\frac{\pi(a)}{\mu(a)} - c\right)_+ \mu(a)(Q^\pi(a) - V)\nabla \log \pi(a).$$

The differences are: (i) we truncate $1/\mu(A) = \pi(A)/\mu(A) \times 1/\pi(A)$ instead of truncating $\pi(A)/\mu(A)$, which provides an additional variance reduction due to the variance of the LR $\nabla \log \pi(A) = \frac{\nabla\pi(A)}{\pi(A)}$ (since this LR may be large when a low probability action is chosen), and (ii) we use our Q-baseline instead of a V baseline, reducing further the variance of the LR estimate.

In our experiments we compare the different policy gradient estimates $\hat{G}_{\text{ISLR}}$, $\hat{G}_{\text{TISLR}}$, and $\hat{G}_{\beta\text{-LOO}}$ for $\beta = 1$, $\beta = 1/\mu$, $\beta = \min(5, 1/\mu)$.

Remark: In off-policy learning it is very difficult to produce an unbiased sample $R(A)$ of $Q^\pi(A)$ when following another policy μ. This would require using full importance sampling correction along the trajectory. Instead, in our implementation we use the off-policy corrected return computed by the Retrace algorithm, which produces a (biased) estimate of $Q^\pi(A)$ but whose bias vanishes asymptotically, see (Munos et al., 2016).

3. Reactor implementation

3.1. Off-policy past actions handled by Retrace

Since the agent's policy changes with time, we need to take into account that past actions $\{a_s \sim \pi_s\}_{s \leq t}$, which have been stored in memory, have been generated according to a policy which is different from the policy π_t we wish to evaluate at current time t. As described in Section 2.1, the Retrace algorithm makes use of two policies, the behaviour and target policies. In Reactor, the behaviour policy μ corresponds to the actions $\{a_s\}_{s \leq t}$ that have been generate by the agent in the past and have been stored in the memory buffer. The target policy corresponds to the current policy π_t of the agent, which is the one we wish to evaluate.

3.2. Behaviour probability estimates $\hat{\mu}$

The agent's behaviour is defined by sampling actions from the current policy π_t and stores observed states, actions, re-

wards and selected action probabilities (x_t, a_t, r_t, μ_t) into the replay memory. Retrace uses an off-policy correction which requires the knowledge of the probability under which actions have been generated (i.e. the trace cutting coefficient c_s in (2) depends on $\mu(a_s|x_s)$). We can either use stored in the memory the behaviour probabilities $\mu(a_s|x_s)$ of the chosen actions or we can construct and estimate $\hat{\mu}$ based on past samples by training to predict previously taken actions $P(a_s|x_s)$. The replay memory contains a mixture of trajectories generated by the sequence of policies $\pi_1, \pi_2, ..., \pi_t$ during the learning process. To see the difference between μ and $\hat{\mu}$, consider that from a state x, actions 1 and 2 have been chosen in a deterministic way with same proportion. Thus the replay memory contains equal amounts of samples for both actions (in that state). From the off-policy batched learning perspective, from that state, the empirical distribution of actions is similar to that of a stochastic policy which would have chosen both actions with equal probabilities of $1/2$. Thus the memorized probabilities $\mu(a_s|x)$ are either 1 or 0, whereas a learnt $\hat{\mu}(a_s|x)$ will predict $1/2$. It can be beneficial to use $\hat{\mu}$ since the trace cutting coefficients c_t would be higher leading to less cutting and faster reward propagation by Retrace algorithm. This might make off-policy corrected returns less biased but at the cost of a higher variance. On the other hand if behavioural probability estimates are inaccurate, Retrace algorithm may produce biased returns.

In the experimental section we report experiments for both approaches. Learnt estimate $\hat{\mu}$ may improve performance compared to using the true action probabilities in some circumstances, which is an interesting finding since those probabilities may not always be available (such as when learning from offline log data). Note that some theoretical findings (Li et al., 2015) support that using a regression estimate may indeed improve importance sampling.

3.3. Network architecture

The Reactor architecture uses a recurrent neural network architecture which takes an observation x_t as input and produces three outputs: current action-value estimates $Q(x_t, a)$, current policy $\pi(a|x_t)$ and estimated behavioural policy $\hat{\mu}(a|x_t)$, for all actions a (Figure 1).

Action-values use a duelling architecture (Wang et al., 2015) which internally splits Q-value into state-value and advantage estimation, which in turn is connected to an LSTM recurrent network (Hochreiter & Schmidhuber, 1997). Policy and behaviour policy heads use last softmax layer mixed with a fixed uniform distribution of choosing a random action where this mixing ratio is a hyperparameter. Each policy network $\hat{\mu}$ and π have their own separate LSTM, as this was found to work much better than sharing the same LSTM with action-value head Q.

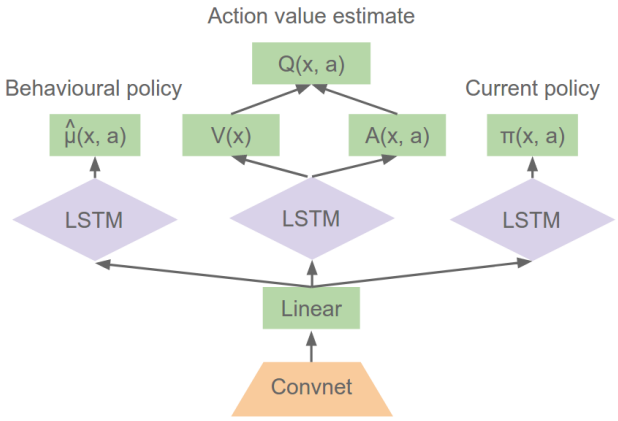

Figure 1. The Reactor architecture.

All three LSTMs in turn are connected to a shared linear layer which is connected to a shared convolutional neural network (Krizhevsky et al., 2012).

Gradients coming from both policy LSTMs are blocked and only gradients originating from Q value LSTM is allowed to back-propagate to the convolutional neural network. We block gradients from π head for increasing stability, as this avoids positive feedback loops between π and Q caused by shared representations. Gradients originating from the $\hat{\mu}$ LSTM are also blocked in order to make experiments with memorized μ and estimated $\hat{\mu}$ values more comparable.

Concatenated rectified linear units are used as nonlinearities in the network (Shang et al., 2016). We did this for two reasons. Firstly, as was shown by (Shang et al., 2016), convolutional neural networks with rectified linear units tend to lead to pairs of opposite features. Using concatenated rectified linear units allows to reduce the size of the filters by half, saving computation. Secondly, neural networks with rectified linear units tend to age, when a given unit gets pushed below zero by a wild gradient and being clamped at zero there is no gradient to ever pull it back up again. Using concatenated rectified linear units helps to solve this problem as if a neuron gets pushed below zero, a value pops up again on the negative side. It is up to the next linearity to learn not to use a value if it is not necessary.

Finally, although we have chosen a recurrent (LSTM) network, a feed-forward implementation is still possible. However in some problems (such as in the Atari domain) it is often useful to base our decisions on a short history of past observations. We chose a recurrent network architecture instead of frame stacking for the reasons of implementation simplicity and computational efficiency. As Re-

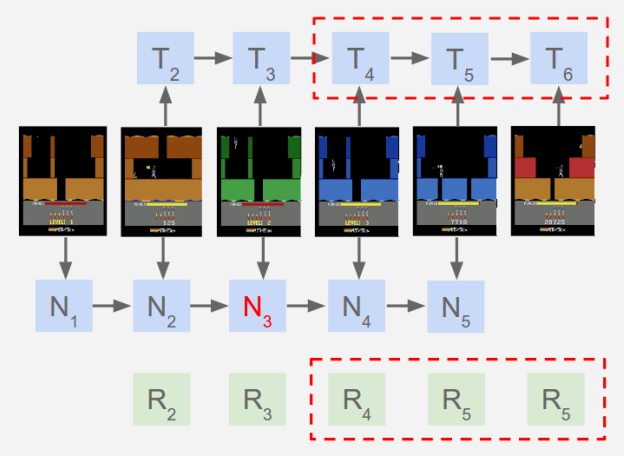

Figure 2. Unrolling current and target networks. Target network **T** is fixed while current network **N** is being trained. The target network is unrolled over a window of sequences offset by one compared to the current network. A new target network Q value estimate (shown in red) depend on target network values and rewards surrounded by red dashed rectangles.

trace algorithm requires evaluating Q-values over contiguous sequences of trajectories, using a recurrent architecture allowed each frame to be processed by the convolutional network only once, as opposed to several times if several frame concatenations were used.

Precise network specification is given in Table 1. The value of ϵ is the minimum probability of choosing a random action and it is hard-coded into the policy network.

3.4. Target networks

We used a target network Q_T (fixed copy of the current network) similar to that of DQN (Mnih et al., 2015). Our implementation of the Retrace algorithm makes use of the target network Q_T. Contiguous sequences $\{x_u, \dots, x_{u+\tau}\}$ of length τ are sampled uniformly at random from our memory buffer. We use the target network Q_T to update our current Q-estimate $Q(x_t, a_t)$ at every state x_t ($t \in [u, u+\tau]$) in the direction of the new target value:

$$r_t + \gamma \mathbb{E}_\pi[Q_T(x_{t+1}, \cdot)] + \sum_{s=t+1}^{u+\tau} \gamma^{s-t}(c_{t+1} \dots c_s)\delta_s^\pi Q_T, \quad (7)$$

where $\delta_s^\pi Q_T = r_s + \gamma \mathbb{E}_\pi[Q_T(x_{s+1}, \cdot)] - Q_T(x_s, a_s)$.

The target network is updated every $T_{update} \in \{1000, 10000\}$ learning steps. We trained all neural networks by sampling contiguous sequences of length $\tau = 33$ and during learning time we always reset the LSTM internal state. In order not to unroll target network over longer

Table 1. Specification of the neural network used.

LAYER	INPUT SIZE	PARAMETERS		
CONVOLUTIONAL		KERNEL WIDTH	OUTPUT CHANNELS	STRIDES
CONV 1	[84, 84, 1]	[8, 8]	16	4
CONCATRELU	[20, 20, 16]			
CONV 2	[20, 20, 32]	[4, 4]	32	2
CONCATRELU	[9, 9, 32]			
CONV 3	[9, 9, 64]	[3, 3]	32	1
CONCATRELU	[7, 7, 32]			
FULLY CONNECTED		OUTPUT SIZE		
LINEAR	[7, 7, 64]	128		
CONCATRELU	[128]			
RECURRENT π AND $\hat{\mu}$		OUTPUT SIZE		
LSTM	[256]	128		
LINEAR	[128]	32		
CONCATRELU	[32]			
LINEAR	[64]	#ACTIONS		
SOFTMAX	[#ACTIONS]	#ACTIONS		
$\chi(1-\epsilon)+\epsilon$/#ACTIONS	[#ACTIONS]	#ACTIONS		
RECURRENT Q		OUTPUT SIZE		
LSTM	[256]	128		
VALUE HEAD		OUTPUT SIZE		
LINEAR	[128]	32		
CONCATRELU	[32]			
LINEAR	[64]	1		
ADVANTAGE HEAD		OUTPUT SIZE		
LINEAR	[128]	32		
CONCATRELU	[32]			
LINEAR	[64]	#ACTIONS		

distances than it was trained on, we evaluated target Q values on sub-sequences of 32 shifted by 1 comparing to the current learning network (Figure 2).

When the agent was acting, the internal state of its policy LSTM was reset only at the beginning of each episode.

3.5. Learning Q, π, $\hat{\mu}$, from a replayed sequence

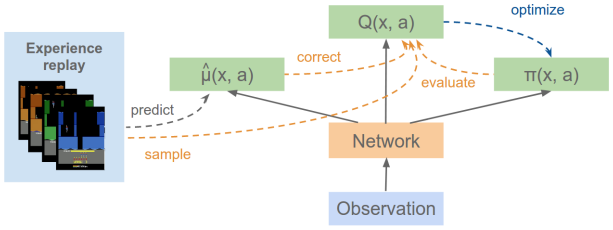

Figure 3. Relationship between different network outputs. $\hat{\mu}$ are estimated behavioural probabilities, Q are estimation of the value function Q^π, and π is the current agent's policy. Q is learnt using the Retrace algorithm. π is learnt using the β-LOO policy gradient algorithm, and $\hat{\mu}$ is trained by supervised learning to predict past actions given past states.

In each learning step, a sequence of length τ is sampled (uniformly) at random from the memory. The target Q_T values are evaluated along the sequence and the current policy π and behaviour policy $\hat{\mu}$ probabilities are evaluated by the current network.

Learning Q-values: Retrace algorithm is used to regress

(using an $L2$-loss) all current Q-values along the sequence towards the new target-values defined by (7). If $\hat{\mu}$ values are used they are evaluated from the moving network in order to get the most up-to-date behavioural estimates. **Learning π:** the current policy π is updated by using the β-LOO policy gradient estimate discussed earlier. In order to avoid converging too quickly to deterministic policies we add a scaled entropy reward to the policy gradient. In order to make entropy reward scale to be more uniform across games with different reward amplitudes and frequencies, during each learning step we evaluated a mean action-gap over a batch of sampled sequences and used it to scale entropy. We defined a mean action-gap as a mean difference between the best action value and a (uniformly) randomly chosen other action. We used 0.1 as a scaling constant for entropy reward. **Learning $\hat{\mu}$:** We regress $\hat{\mu}$ towards actions sampled from replay memory using a cross entropy loss.

We summed all losses, evaluated gradients and used ADAM optimizer (Kingma & Ba, 2014) to train the neural network.

3.6. Parallelism

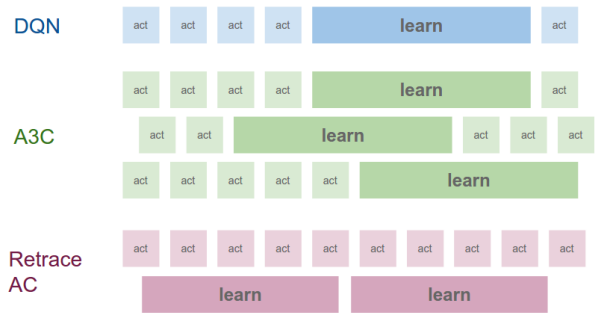

Figure 4. The model of parallelism. Reactor uses a single actor thread acting on the environment and a single learner thread training the neural network by sampling batched trajectories from the neural network. Both threads run asynchronously. This is different than the approach used by DQN (a single thread alternating acting and learning) and an A3C (multiple asynchronous threads alternating learning and acting).

In order to improve CPU/GPU utilization we can decouple acting from learning. This is an important aspect of our architecture: an *acting thread* receives observations, submits actions to the environment, and stores transition information into the memory, while an *learning thread* resamples sequences of experiences from the memory and learns from them (Figure 4). Comparing to the standard DQN framework (Mnih et al., 2015) Reactor is able to improve CPU/GPU utilization by learning while acting, and comparing to A3C framework (Mnih et al., 2016) we can exploit computational benefits of batched learning.

4. Results

4.1. Comparison of different policy gradients within Reactor architecture

We trained different versions of Reactor on 57 Atari games for 200 million frames with 3 randomly initialized seeds per experiment. We used the following method to compare algorithm performance across all games as a function of training time: (1). For each game for each algorithm pair and for each time step we evaluated the probability that algorithm 1 has more score than algorithm 2 by comparing all seeds pair-wise. This produced a tensor of size P[game x alg1 x alg2 x time]. (2). For each algorithm pair and for each time-step we averaged probabilities across games producing Q[alg1 x alg2 x time]. This quantity has an interpretable meaning of the probability that a random instance of algorithm 1 would be outperforming a random instance of algorithm 2 at time t given a randomly chosen game. (3). For each algorithm and each time we evaluated the average probability by averaging across all other algorithms. This produced a matrix S[alg x time] which has a meaning of a probabiliy that an algorithm outperforms any other randomly selected algorithm on a randomly selected game at training step t. We plot those curves as a function of time in order to compare relative algorithm performance at different stages of learning.

4.2. Estimating versus memorizing behavioural probabilities

We evaluated inter-game performance metric described in section 4.1. Using the slowest target network update frequency $T_{update} = 10000$ gave better results than using $T_{update} = 1000$ and using a lower value of $\lambda = 0.9$ gave better results than $\lambda = 1.0$. The results are shown in Figures 5 and 6. As it is seen, memorizing behavioural probability values on average gave better results for most algorithms and most parameter configurations that when using estimated values, especially for target network update period $T_{update} = 10000$. A notable exception is TISLR algorithm, where estimated behavioural probabilities gave better scores over most training steps and most parameter configurations. Still, a difference in performance was not so huge, as at 200 million steps learning behavioural probabilities had roughly 1/3 chance of giving better results than memorizing values for both beta-LOO algorithms and gave similar performance in the case of TSILR (Figure 5). This demonstrates that estimating behavioural probabilities is a feasible approach when actual behaviour probability values are not available, which could happen when learning from offline log data or in the case of apprentice learning.

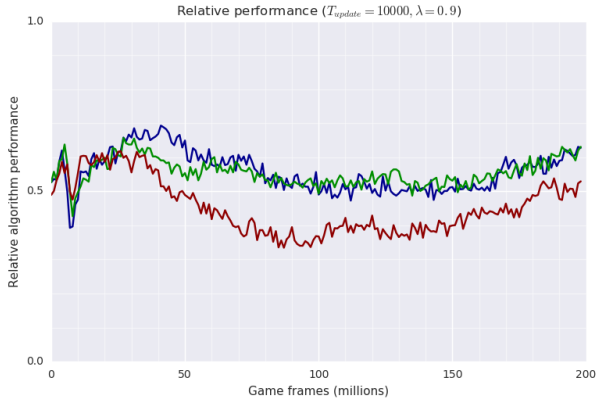

Figure 5. Comparison of a relative performance of memorizing versus learning behaviour probability values for three different tested policy gradient algorithms as a function of training epoch and the best found parameter configuration of T_{update} = 10000, $\lambda = 0.9$. Y axis represents a probability that a given policy gradient algorithm would score more when memorized behaviour probability μ values are used than when estimating behaviour probability $\hat{\mu}$ values are used on a randomly selected Atari game at a given training epoch. Each training epoch consisted of 1 million environment steps. Each algorithm was run with three random seeds and the curves include a comparison of all pairs of seeds.

4.3. Comparing different policy gradients

As can be seen from Figure 6, leave-one-out with $\beta = 1$ performed slightly better than leave-one-out with $\beta = min(1/\mu, 5)$ for both memorized and estimated behaviour probability values. Both algorithms performed better than TISLR for memorized behavioural probabilities and performed similarly when probabilities were estimated.

4.4. Comparing different architectures

In order to compare different algorithms, we evaluated the best Reactor version obtained from Figure 6 by evaluating it on 200 episodes with 30 random human starts on each Atari game. We evaluated mean episode rewards for each atari game and human normalized scores. We calculated mean and median human normalized scores across all games. In addition to that, we ranked all algorithms (including random and human scores) for each game and evaluated mean rank of each algorithm across all 57 Atari games. We believe that this is a better metric to compare algorithms across different games with vastly different rewards scales because it is less sensitive to the errors introduced when measuring human play, which otherwise is present in the denominator when calculating human normalized scores. We also evaluated an Elo scores for each algorithm, where algorithm A is considered to win over algorithm B if it collects more score on a given Atari game when using 30 random human starts. In our Elo evaluation difference in ranking 400 correspond to the odds of winning of 10:1.

Table 2. Comparison of algorithm performance across 57 Atari games with 30 random human starts. Human Elo was defined to be zero. Value above zero indicate super-human performance. Reactor was trained with $\lambda = 1.0$ and target network update period of 10^4 learning steps.

ALGORITHM	NORMALIZED SCORES MEAN	NORMALIZED SCORES MEDIAN	RANK MEAN	ELO
Across all games				
RANDOM	0.00	0.00	7.67	-520
HUMAN	1.00	1.00	4.19	0
DQN	2.17	0.69	5.61	-153
DDQN	3.31	1.11	4.42	-24
DUEL	3.42	1.17	3.49	71
PRIOR	3.84	1.13	3.77	43
PRIOR. DUEL.	**5.65**	1.15	3.47	73
REACTOR M1	5.55	**1.40**	**3.40**	**81**

Table 3. Comparison of algorithm performance across 57 Atari games with 30 random noop starts. We reported the final performance by taking the best training curve value from each learning curve in order to make evaluation comparable to the results reported by (Wang et al., 2017). Experiments marked with letter M indicate memorized μ while L indicates learnt value of $\hat{\mu}$. Algorithms L1 and M1 correspond to β-LOO with $\beta = 1$ while L5 and M5 correspond to β-LOO with $\beta = \min(1/\hat{\mu}, 5)$ and $\beta = \min(1/\mu, 5)$ respectively.

ALGORITHM	NORMALIZED SCORES	MEAN RANK	ELO
RANDOM	0.00	8.00	-808
HUMAN	1.00	4.74	0
ACER	1.55	-	-
REACTOR M5	1.40	3.81	100
REACTOR M1	1.51	**3.42**	**139**
REACTOR M TISLR	1.28	4.82	-2
REACTOR L5	1.54	4.12	71
REACTOR L1	1.61	3.84	99
REACTOR L TISLR	**1.72**	3.65	116

Table 2 contains a comparison of our algorithm with several other synchronous state-of-art algorithms across 57 Atari games for a fixed random seed across all games (Bellemare et al., 2013). The algorithms that we compare Reactor to are: DQN (Mnih et al., 2015), Double DQN (Van Hasselt et al., 2016), DQN with prioritized experience replay (Schaul et al., 2015), duelling architecture and prioritized duelling architecture (Wang et al., 2015). Each algorithm was exposed to 200 million frames of experience and the

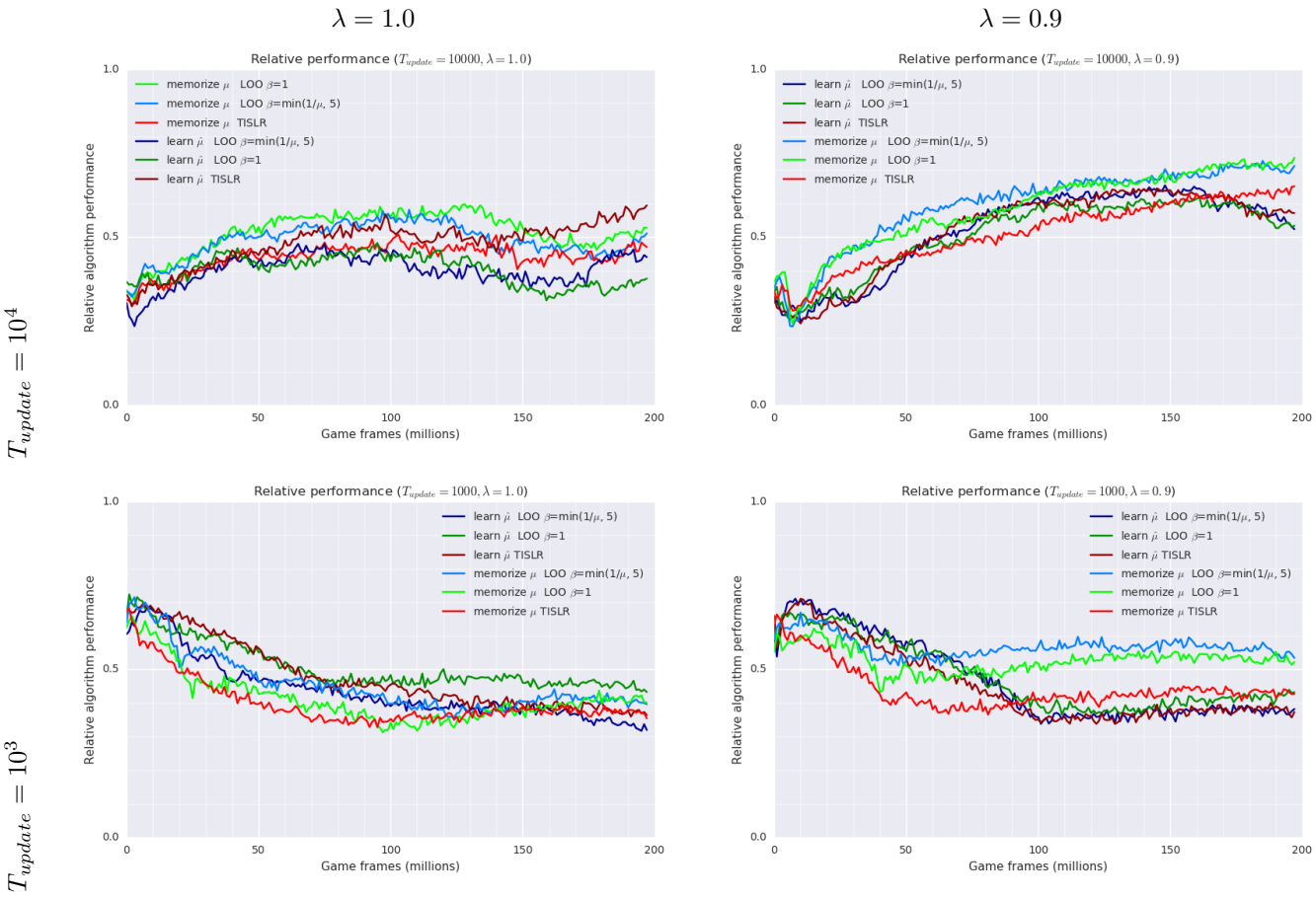

Figure 6. Comparison of performance of different policy gradient algorithms with memorized μ_t and estimated $\hat{\mu}_t$ values of behavioural probabilities. Darker colours correspond to learnt values $\hat{\mu}_t$ while lighter curves correspond to memorized probabilities μ_t. Y axis represents a probability that a given algorithm would score more on a randomly selected algorithm with one of three randomly selected T_{update} and λ configurations on a randomly selected Atari game at a training given epoch. Each training epoch consisted of 1 million environment steps.

same pre-processing pipeline including 4 action repeats was used as in the original DQN paper (Mnih et al., 2015). We tested the version of Reactor which memorized past behaviour probabilities and used $\beta = 1$. Reactor was able to improve over prioritized duelling agent in terms of median human normalized scores, mean rank and Elo, but was slightly worse in terms of mean human normalized scores.

Table 3 contains a comparison of our algorithm with ACER algorithm (Wang et al., 2017). In the case of Reactor, the table was produced in the following way. While the agent was training on each game, scores were averaged over 200 windows of length 1 million. The maximum value was taken from each curve and it was human-normalized. A median value was evaluated for each algorithm across all games. The best value for ACER was obtained from Figure 1 in (Wang et al., 2017) by reading off the highest learning curve below 200 million steps. As it is seen from the results two versions of Reactor improved over the ACER algorithm in terms of median human normalized scores. On the other hand our evaluation was somewhat more pessimistic than the one used by (Wang et al., 2017), because we averaged rewards over windows of 1 million steps, while in the case of ACER the rewards were averaged over windows of 200 thousand frames. Averaging over longer windows leads to less variance, which in turn introduces less optimistic bias encountered during the maximization process. We were not able to evaluate Rank and Elo scores for ACER because we did not have per-game evaluations. One interesting finding was that our implementation of TISLR algorithm with learnt probabilities achieved better results than the ones reported in the ACER paper when using learn behavioural probabilities.

5. Conclusion

In this work we presented a new off-policy agent based on Retrace actor critic architecture and demonstrated that it can achieve similar performance as the current state of the art. We also demonstrated that estimated behavioural probabilities can outperformed memorized behavioural probabilities under some circumstances when used for off-policy correction. As the framework is fully off-policy, it can be used to test different exploration ideas.

References

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, 2013.

Harb, Jean and Precup, Doina. Investigating recurrence and eligibility traces in deep q-networks.

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

Jaderberg, Max, Mnih, Volodymyr, Czarnecki, Wojciech Marian, Schaul, Tom, Leibo, Joel Z, Silver, David, and Kavukcuoglu, Koray. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Li, Lihong, Munos, Rémi, and Szepesvári, Csaba. Toward minimax off-policy value estimation. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, 2015.

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, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 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.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Shang, Wenling, Sohn, Kihyuk, Almeida, Diogo, and Lee, Honglak. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Sutton, Richard S., Mcallester, David, Singh, Satinder, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In *In Advances in Neural Information Processing Systems 12*, pp. 1057–1063. MIT Press, 2000.

Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100, 2016.

Wang, Ziyu, Schaul, Tom, Hessel, Matteo, van Hasselt, Hado, Lanctot, Marc, and de Freitas, Nando. Dueling network architectures for deep reinforcement learning. *International Conference on Machine Learning*, pp. 1995–2003, 2015.

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.

6. Appendix

Table 4. Scores for each game evaluated with 30 random human starts. Reactor was evaluated by averaging scores over 200 episodes while other algorithms were evaluated by averaging scores over 100 episodes. All scores (except for Reactor) were taken from (Wang et al., 2015)

.

GAME AGENT	RANDOM	HUMAN	DQN	DDQN	DUEL	PRIOR	PRIOR. DUEL.	REACTOR M1
ALIEN	128.3	**6371.3**	634.0	1033.4	1486.5	1334.7	823.7	856.5
AMIDAR	11.8	**1540.4**	178.4	169.1	172.7	129.1	238.4	221.4
ASSAULT	166.9	628.9	3489.3	6060.8	3994.8	6548.9	10950.6	**15511.3**
ASTERIX	164.5	7536.0	3170.5	16837.0	15840.0	22484.5	**364200.0**	16797.0
ASTEROIDS	871.3	**36517.3**	1458.7	1193.2	2035.4	1745.1	1021.9	6207.4
ATLANTIS	13463.0	26575.0	292491.0	319688.0	445360.0	330647.0	423252.0	**705930.5**
BANK HEIST	21.7	644.5	312.7	886.0	**1129.3**	876.6	1004.6	901.2
BATTLEZONE	3560.0	33030.0	23750.0	24740.0	31320.0	25520.0	30650.0	**76385.0**
BEAM RIDER	254.6	14961.0	9743.2	17417.2	14591.3	31181.3	**37412.2**	6994.8
BERZERK	196.1	**2237.5**	493.4	1011.1	910.6	865.9	2178.6	912.5
BOWLING	35.2	**146.5**	56.5	69.6	65.7	52.0	50.4	70.3
BOXING	-1.5	9.6	70.3	73.5	77.3	72.3	**79.2**	68.2
BREAKOUT	1.6	27.9	354.5	368.9	411.6	343.0	354.6	**457.3**
CENTIPEDE	1925.5	**10321.9**	3973.9	3853.5	4881.0	3489.1	5570.2	3522.4
CHOPPER COMMAND	644.0	**8930.0**	5017.0	3495.0	3784.0	4635.0	8058.0	5080.0
CRAZY CLIMBER	9337.0	32667.0	98128.0	113782.0	124566.0	127512.0	127853.0	**144593.5**
DEFENDER	1965.5	14296.0	15917.5	27510.0	33996.0	23666.5	34415.0	**50490.5**
DEMON ATTACK	208.3	3442.8	12550.7	69803.4	56322.8	61277.5	**73371.3**	45122.3
DOUBLE DUNK	-16.0	-14.4	-6.0	-0.3	-0.8	16.0	-10.7	**21.5**
ENDURO	-81.8	740.2	626.7	1216.6	2077.4	1831.0	**2223.9**	2110.1
FISHING DERBY	-77.1	5.1	-1.6	3.2	-4.1	9.8	**17.0**	16.0
FREEWAY	0.1	25.6	26.9	28.8	0.2	28.9	28.2	**29.2**
FROSTBITE	66.4	**4202.8**	496.1	1448.1	2332.4	3510.0	4038.4	3348.9
GOPHER	250.0	2311.0	8190.4	15253.0	20051.4	34858.8	**105148.4**	60018.6
GRAVITAR	245.5	**3116.0**	298.0	200.5	297.0	269.5	167.0	497.2
H.E.R.O.	1580.3	**25839.4**	14992.9	14892.5	15207.9	20889.9	15459.2	10604.0
ICE HOCKEY	-9.7	0.5	-1.6	-2.5	-1.3	-0.2	0.5	**5.4**
JAMES BOND 007	33.5	368.5	697.5	573.0	835.5	**3961.0**	585.0	3049.0
KANGAROO	100.0	2739.0	4496.0	11204.0	10334.0	**12185.0**	861.0	7163.5
KRULL	1151.9	2109.1	6206.0	6796.1	8051.6	6872.8	7658.6	**8148.6**
KUNG-FU MASTER	304.0	20786.8	20882.0	30207.0	24288.0	31676.0	37484.0	**46172.5**
MONTEZUMA'S REVENGE	25.0	**4182.0**	47.0	42.0	22.0	51.0	24.0	23.0
MS. PAC-MAN	197.8	**15375.0**	1092.3	1241.3	2250.6	1865.9	1007.8	1207.5
NAME THIS GAME	1747.8	6796.0	6738.8	8960.3	11185.1	10497.6	**13637.9**	11897.0
PHOENIX	1134.4	6686.2	7484.8	12366.5	20410.5	16903.6	**63597.0**	5844.1
PITFALL!	-348.8	**5998.9**	-113.2	-186.7	-46.9	-427.0	-243.6	-198.4
PONG	-18.0	15.5	18.0	**19.1**	18.8	18.9	18.4	18.9
PRIVATE EYE	662.8	**64169.1**	207.9	-575.5	292.6	670.7	1277.6	-161.4
Q*BERT	183.0	12085.0	9271.5	11020.8	**14175.8**	9944.0	14063.0	7413.5
RIVER RAID	588.3	14382.2	4748.5	10838.4	**16569.4**	11807.2	16496.8	6113.6
ROAD RUNNER	200.0	6878.0	35215.0	43156.0	**58549.0**	52264.0	54630.0	49709.0
ROBOTANK	2.4	8.9	58.7	59.1	**62.0**	56.2	24.7	59.3
SEAQUEST	215.5	**40425.8**	4216.7	14498.0	37361.6	25463.7	1431.2	5943.3
SKIING	-15287.4	**-3686.6**	-12142.1	-11490.4	-11928.0	-10169.1	-18955.8	-15464.8
SOLARIS	2047.2	**11032.6**	1295.4	810.0	1768.4	2272.8	280.6	840.2
SPACE INVADERS	182.6	1464.9	1293.8	2628.7	5993.1	3912.1	**8978.0**	1704.8
STARGUNNER	697.0	9528.0	52970.0	58365.0	90804.0	61582.0	**127073.0**	45829.0
SURROUND	-9.7	5.4	-6.0	1.9	4.0	**5.9**	-0.2	-0.9
TENNIS	-21.4	-6.7	11.1	-7.8	4.4	-5.3	-13.2	**22.5**
TIME PILOT	3273.0	5650.0	4786.0	6608.0	6601.0	5963.0	4871.0	**14596.0**
TUTANKHAM	12.7	**138.3**	45.6	92.2	48.0	56.9	108.6	130.5
UP'N DOWN	707.2	9896.1	8038.5	19086.9	24759.2	12157.4	22681.3	**165934.0**
VENTURE	18.0	**1039.0**	136.0	21.0	200.0	94.0	29.0	7.5
VIDEO PINBALL	20452.0	15641.1	154414.1	367823.7	110976.2	295972.8	447408.6	**550035.6**
WIZARD OF WOR	804.0	4556.0	1609.0	6201.0	7054.0	5727.0	**10471.0**	8658.0
YARS' REVENGE	1476.9	47135.2	4577.5	6270.6	25976.5	4687.4	58145.9	**65248.1**
ZAXXON	475.0	8443.0	4412.0	8593.0	10164.0	9474.0	11320.0	**11980.0**

Table 5. Human normalized scores obtained by evaluating agents with 30 random human starts.

.

GAME AGENT	RANDOM	HUMAN	DQN	DDQN	DUEL	PRIOR	PRIOR. DUEL.	REACTOR M1
ALIEN	0.00	**1.00**	0.08	0.14	0.22	0.19	0.11	0.12
AMIDAR	0.00	**1.00**	0.11	0.10	0.11	0.08	0.15	0.14
ASSAULT	0.00	1.00	7.19	12.76	8.29	13.81	23.34	**33.21**
ASTERIX	0.00	1.00	0.41	2.26	2.13	3.03	**49.38**	2.26
ASTEROIDS	0.00	**1.00**	0.02	0.01	0.03	0.02	0.00	0.15
ATLANTIS	0.00	1.00	21.28	23.35	32.94	24.19	31.25	**52.81**
BANK HEIST	0.00	1.00	0.47	1.39	**1.78**	1.37	1.58	1.41
BATTLEZONE	0.00	1.00	0.69	0.72	0.94	0.75	0.92	**2.47**
BEAM RIDER	0.00	1.00	0.65	1.17	0.97	2.10	**2.53**	0.46
BERZERK	0.00	**1.00**	0.15	0.40	0.35	0.33	0.97	0.35
BOWLING	0.00	**1.00**	0.19	0.31	0.27	0.15	0.14	0.32
BOXING	0.00	1.00	6.47	6.76	7.10	6.65	**7.27**	6.28
BREAKOUT	0.00	1.00	13.42	13.97	15.59	12.98	13.42	**17.33**
CENTIPEDE	0.00	**1.00**	0.24	0.23	0.35	0.19	0.43	0.19
CHOPPER COMMAND	0.00	**1.00**	0.53	0.34	0.38	0.48	0.89	0.54
CRAZY CLIMBER	0.00	1.00	3.81	4.48	4.94	5.07	5.08	**5.80**
DEFENDER	0.00	1.00	1.13	2.07	2.60	1.76	2.63	**3.94**
DEMON ATTACK	0.00	1.00	3.82	21.52	17.35	18.88	**22.62**	13.89
DOUBLE DUNK	0.00	1.00	6.25	9.81	9.50	20.00	3.31	**23.44**
ENDURO	0.00	1.00	0.86	1.58	2.63	2.33	**2.80**	2.67
FISHING DERBY	0.00	1.00	0.92	0.98	0.89	1.06	**1.14**	1.13
FREEWAY	0.00	1.00	1.05	1.13	0.00	1.13	1.10	**1.14**
FROSTBITE	0.00	**1.00**	0.10	0.33	0.55	0.83	0.96	0.79
GOPHER	0.00	1.00	3.85	7.28	9.61	16.79	**50.90**	29.00
GRAVITAR	0.00	**1.00**	0.02	-0.02	0.02	0.01	-0.03	0.09
H.E.R.O.	0.00	**1.00**	0.55	0.55	0.56	0.80	0.57	0.37
ICE HOCKEY	0.00	1.00	0.79	0.71	0.82	0.93	1.00	**1.48**
JAMES BOND 007	0.00	1.00	1.98	1.61	2.39	**11.72**	1.65	9.00
KANGAROO	0.00	1.00	1.67	4.21	3.88	**4.58**	0.29	2.68
KRULL	0.00	1.00	5.28	5.90	7.21	5.98	6.80	**7.31**
KUNG-FU MASTER	0.00	1.00	1.00	1.46	1.17	1.53	1.82	**2.24**
MONTEZUMA'S REVENGE	0.00	**1.00**	0.01	0.00	-0.00	0.01	-0.00	-0.00
MS. PAC-MAN	0.00	**1.00**	0.06	0.07	0.14	0.11	0.05	0.07
NAME THIS GAME	0.00	1.00	0.99	1.43	1.87	1.73	**2.36**	2.01
PHOENIX	0.00	1.00	1.14	2.02	3.47	2.84	**11.25**	0.85
PITFALL!	0.00	**1.00**	0.04	0.03	0.05	-0.01	0.02	0.02
PONG	0.00	1.00	1.07	**1.11**	1.10	1.10	1.09	1.10
PRIVATE EYE	0.00	**1.00**	-0.01	-0.02	-0.01	0.00	0.01	-0.01
Q*BERT	0.00	1.00	0.76	0.91	**1.18**	0.82	1.17	0.61
RIVER RAID	0.00	1.00	0.30	0.74	**1.16**	0.81	1.15	0.40
ROAD RUNNER	0.00	1.00	5.24	6.43	**8.74**	7.80	8.15	7.41
ROBOTANK	0.00	1.00	8.66	8.72	**9.17**	8.28	3.43	8.75
SEAQUEST	0.00	**1.00**	0.10	0.36	0.92	0.63	0.03	0.14
SKIING	0.00	**1.00**	0.27	0.33	0.29	0.44	-0.32	-0.02
SOLARIS	0.00	**1.00**	-0.08	-0.14	-0.03	0.03	-0.20	-0.13
SPACE INVADERS	0.00	1.00	0.87	1.91	4.53	2.91	**6.86**	1.19
STARGUNNER	0.00	1.00	5.92	6.53	10.20	6.89	**14.31**	5.11
SURROUND	0.00	1.00	0.25	0.77	0.91	**1.03**	0.63	0.58
TENNIS	0.00	1.00	2.21	0.93	1.76	1.10	0.56	**2.99**
TIME PILOT	0.00	1.00	0.64	1.40	1.40	1.13	0.67	**4.76**
TUTANKHAM	0.00	**1.00**	0.26	0.63	0.28	0.35	0.76	0.94
UP'N DOWN	0.00	1.00	0.80	2.00	2.62	1.25	2.39	**17.98**
VENTURE	0.00	**1.00**	0.12	0.00	0.18	0.07	0.01	-0.01
VIDEO PINBALL	0.00	1.00	8.56	22.21	5.79	17.62	27.30	**33.86**
WIZARD OF WOR	0.00	1.00	0.21	1.44	1.67	1.31	**2.58**	2.09
YARS' REVENGE	0.00	1.00	0.07	0.10	0.54	0.07	1.24	**1.40**
ZAXXON	0.00	1.00	0.49	1.02	1.22	1.13	1.36	**1.44**