The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver
DeepMind

Abstract

The deep reinforcement learning community has made several independent improvements to the DQN algorithm. However, it is unclear which of these extensions are complementary and can be fruitfully combined. This paper examines six extensions to the DQN algorithm and empirically studies their combination. Our experiments show that the combination provides state-of-the-art performance on the Atari 2600 benchmark, both in terms of data efficiency and final performance. We also provide results from a detailed ablation study that shows the contribution of each component to overall performance.

Introduction

The many recent successes in scaling reinforcement learning (RL) to complex sequential decision-making problems were kick-started by the Deep Q-Networks algorithm (DQN; Mnih et al. 2013, 2015). Its combination of Q-learning with convolutional neural networks and experience replay enabled it to learn, from raw pixels, how to play many Atari games at human-level performance. Since then, many extensions have been proposed that enhance its speed or stability.

Double DQN (DDQN; van Hasselt, Guez, and Silver 2016) addresses an overestimation bias of Q-learning (van Hasselt 2010), by decoupling selection and evaluation of the bootstrap action. Prioritized experience replay (Schaul et al. 2015) improves data efficiency, by replaying more often transitions from which there is more to learn. The dueling network architecture (Wang et al. 2016) helps to generalize across actions by separately representing state values and action advantages. Learning from multi-step bootstrap targets (Sutton 1988; Sutton and Barto 1998), as used in A3C (Mnih et al. 2016), shifts the bias-variance tradeoff and helps to propagate newly observed rewards faster to earlier visited states. Distributional Q-learning (Bellemare, Dabney, and Munos 2017) learns a categorical distribution of discounted returns, instead of estimating the mean. Noisy DQN (Fortunato et al. 2017) uses stochastic network layers for exploration. This list is, of course, far from exhaustive.

Each of these algorithms enables substantial performance improvements in isolation. Since they address radically different issues, and since they build on a shared framework, they could plausibly be combined. In some cases this has been done: Prioritized DDQN and Dueling DDQN both use double Q-learning, and Dueling DDQN was also combined with prioritized replay. In this paper we propose to study an agent that combines all the aforementioned ingredients. We show how these different ideas can be integrated, and that they are indeed complementary. In fact, their combination results in new state-of-the-art results on the benchmark suite of 57 Atari 2600 games from the Arcade Learning Environment (Bellemare et al. 2013), both in terms of data efficiency and of final performance. Finally, we show results from ablation studies to help understand the contributions of the individual components.

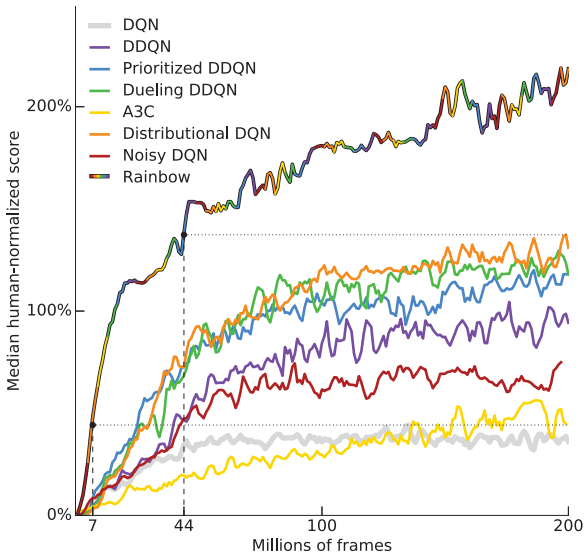

Figure 1: Median human-normalized performance across 57 Atari games. We compare Rainbow (rainbow-colored) to DQN and six published baselines. We match DQN's best performance after 7M frames, surpass any baseline in 44M frames, reaching substantially improved final performance. Curves are smoothed with a moving average of 5 points.

Background

Reinforcement learning addresses the problem of an *agent* learning to act in an *environment* in order to maximize a scalar *reward* signal. No direct supervision is provided to the agent, for instance it is never directly told the best action.

Agents and environments. At each discrete time step $t = 0, 1, 2 \ldots$, the environment provides the agent with an observation S_t, the agent responds by selecting an action A_t, and then the environment provides the next reward R_{t+1}, discount γ_{t+1}, and state S_{t+1}. This interaction is formalized as a *Markov Decision Process*, or MDP, which is a tuple $\langle \mathcal{S}, \mathcal{A}, T, r, \gamma \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $T(s, a, s') = P[S_{t+1} = s' \mid S_t = s, A_t = a]$ is the (stochastic) transition function, $r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$ is the reward function, and $\gamma \in [0, 1]$ is a discount factor. In our experiments MDPs will be *episodic* with a constant $\gamma_t = \gamma$, except on episode termination where $\gamma_t = 0$, but the algorithms are expressed in the general form.

On the agent side, action selection is given by a policy π that defines a probability distribution over actions for each state. From the state S_t encountered at time t, we define the discounted return $G_t = \sum_{k=0}^{\infty} \gamma_t^{(k)} R_{t+k+1}$ as the discounted sum of future rewards collected by the agent, where the discount for a reward k steps in the future is given by the product of discounts before that time, $\gamma_t^{(k)} = \prod_{i=1}^{k} \gamma_{t+i}$. An agent aims to maximize the expected discounted return by finding a good policy.

The policy may be learned directly, or it may be constructed as a function of some other learned quantities. In value-based reinforcement learning, the agent learns an estimate of the expected discounted return, or value, when following a policy π starting from a given state, $v^\pi(s) = E_\pi[G_t | S_t = s]$, or state-action pair, $q^\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$. A common way of deriving a new policy from a state-action value function is to act ϵ-greedily with respect to the action values. This corresponds to taking the action with the highest value (the *greedy* action) with probability $(1-\epsilon)$, and to otherwise act uniformly at random with probability ϵ. Policies of this kind are used to introduce a form of *exploration*: by randomly selecting actions that are sub-optimal according to its current estimates, the agent can discover and correct its estimates when appropriate. The main limitation is that it is difficult to discover alternative courses of action that extend far into the future; this has motivated research on more directed forms of exploration.

Deep reinforcement learning and DQN. Large state and/or action spaces make it intractable to learn Q value estimates for each state and action pair independently. In deep reinforcement learning, we represent the various components of agents, such as policies $\pi(s, a)$ or values $q(s, a)$, with deep (i.e., multi-layer) neural networks. The parameters of these networks are trained by gradient descent to minimize some suitable loss function.

In DQN (Mnih et al. 2015) deep networks and reinforcement learning were successfully combined by using a convolutional neural net to approximate the action values for a given state S_t (which is fed as input to the network in the form of a stack of raw pixel frames). At each step, based on the current state, the agent selects an action ϵ-greedily with respect to the action values, and adds a transition $(S_t, A_t, R_{t+1}, \gamma_{t+1}, S_{t+1})$ to a replay memory buffer (Lin 1992), that holds the last million transitions. The parameters of the neural network are optimized by using stochastic gradient descent to minimize the loss

$$(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\overline{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t))^2 , \qquad (1)$$

where t is a time step randomly picked from the replay memory. The gradient of the loss is back-propagated only into the parameters θ of the *online network* (which is also used to select actions); the term $\overline{\theta}$ represents the parameters of a *target network*; a periodic copy of the online network which is not directly optimized. The optimization is performed using RMSprop (Tieleman and Hinton 2012), a variant of stochastic gradient descent, on mini-batches sampled uniformly from the experience replay. This means that in the loss above, the time index t will be a random time index from the last million transitions, rather than the current time. The use of experience replay and target networks enables relatively stable learning of Q values, and led to superhuman performance on several Atari games.

Extensions to DQN

DQN has been an important milestone, but several limitations of this algorithm are now known, and many extensions have been proposed. We propose a selection of six extensions that each have addressed a limitation and improved overall performance. To keep the size of the selection manageable, we picked a set of extensions that address distinct concerns (e.g., just one of the many addressing exploration).

Double Q-learning. Conventional Q-learning is affected by an overestimation bias, due to the maximization step in Equation 1, and this can harm learning. Double Q-learning (van Hasselt 2010), addresses this overestimation by decoupling, in the maximization performed for the bootstrap target, the selection of the action from its evaluation. It is possible to effectively combine this with DQN (van Hasselt, Guez, and Silver 2016), using the loss

$$(R_{t+1} + \gamma_{t+1} q_{\overline{\theta}}(S_{t+1}, \operatorname*{argmax}_{a'} q_\theta(S_{t+1}, a')) - q_\theta(S_t, A_t))^2.$$

This change was shown to reduce harmful overestimations that were present for DQN, thereby improving performance.

Prioritized replay. DQN samples uniformly from the replay buffer. Ideally, we want to sample more frequently those transitions from which there is much to learn. As a proxy for learning potential, prioritized experience replay (Schaul et al. 2015) samples transitions with probability p_t relative to the last encountered absolute *TD error*:

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\overline{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t) \right|^\omega ,$$

where ω is a hyper-parameter that determines the shape of the distribution. New transitions are inserted into the replay

buffer with maximum priority, providing a bias towards recent transitions. Note that stochastic transitions might also be favoured, even when there is little left to learn about them.

Dueling networks. The dueling network is a neural network architecture designed for value based RL. It features two streams of computation, the value and advantage streams, sharing a convolutional encoder, and merged by a special aggregator (Wang et al. 2016). This corresponds to the following factorization of action values:

$$q_\theta(s,a) = v_\eta(f_\xi(s)) + a_\psi(f_\xi(s),a) - \frac{\sum_{a'} a_\psi(f_\xi(s),a')}{N_{\text{actions}}},$$

where ξ, η, and ψ are, respectively, the parameters of the shared encoder f_ξ, of the value stream v_η, and of the advantage stream a_ψ; and $\theta = \{\xi, \eta, \psi\}$ is their concatenation.

Multi-step learning. Q-learning accumulates a single reward and then uses the greedy action at the next step to bootstrap. Alternatively, forward-view *multi-step* targets can be used (Sutton 1988). We define the truncated n-step return from a given state S_t as

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}\,. \tag{2}$$

A multi-step variant of DQN is then defined by minimizing the alternative loss,

$$(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_{\overline{\theta}}(S_{t+n}, a') - q_\theta(S_t, A_t))^2.$$

Multi-step targets with suitably tuned n often lead to faster learning (Sutton and Barto 1998).

Distributional RL. We can learn to approximate the distribution of returns instead of the expected return. Recently Bellemare, Dabney, and Munos (2017) proposed to model such distributions with probability masses placed on a discrete support $\boldsymbol{z}$, where $\boldsymbol{z}$ is a vector with $N_{\text{atoms}} \in \mathbb{N}^+$ *atoms*, defined by $z^i = v_{\min} + (i-1)\frac{v_{\max}-v_{\min}}{N_{\text{atoms}}-1}$ for $i \in \{1, \ldots, N_{\text{atoms}}\}$. The approximating distribution d_t at time t is defined on this support, with the probability mass $p_\theta^i(S_t, A_t)$ on each atom i, such that $d_t = (\boldsymbol{z}, \boldsymbol{p}_\theta(S_t, A_t))$. The goal is to update θ such that this distribution closely matches the actual distribution of returns.

To learn the probability masses, the key insight is that return distributions satisfy a variant of Bellman's equation. For a given state S_t and action A_t, the distribution of the returns under the optimal policy π^* should match a target distribution defined by taking the distribution for the next state S_{t+1} and action $a^*_{t+1} = \pi^*(S_{t+1})$, contracting it towards zero according to the discount, and shifting it by the reward (or distribution of rewards, in the stochastic case). A distributional variant of Q-learning is then derived by first constructing a new support for the target distribution, and then minimizing the Kullbeck-Leibler divergence between the distribution d_t and the target distribution $d'_t \equiv (R_{t+1} + \gamma_{t+1}\boldsymbol{z},\ \boldsymbol{p}_{\overline{\theta}}(S_{t+1}, \overline{a}^*_{t+1}))$,

$$D_{\text{KL}}(\Phi_{\boldsymbol{z}} d'_t || d_t)\,. \tag{3}$$

Here $\Phi_{\boldsymbol{z}}$ is a L2-projection of the target distribution onto the fixed support $\boldsymbol{z}$, and $\overline{a}^*_{t+1} = \text{argmax}_a\, q_{\overline{\theta}}(S_{t+1}, a)$ is the greedy action with respect to the mean action values $q_{\overline{\theta}}(S_{t+1}, a) = \boldsymbol{z}^\top \boldsymbol{p}_\theta(S_{t+1}, a)$ in state S_{t+1}.

As in the non-distributional case, we can use a frozen copy of the parameters $\overline{\theta}$ to construct the target distribution. The parametrized distribution can be represented by a neural network, as in DQN, but with $N_{\text{atoms}} \times N_{\text{actions}}$ outputs. A *softmax* is applied independently for each action dimension of the output to ensure that the distribution for each action is appropriately normalized.

Noisy Nets. The limitations of exploring using ϵ-greedy policies are clear in games such as Montezuma's Revenge, where many actions must be executed to collect the first reward. Noisy Nets (Fortunato et al. 2017) propose a noisy linear layer that combines a deterministic and noisy stream,

$$\boldsymbol{y} = (\boldsymbol{b} + \mathbf{W}\boldsymbol{x}) + (\boldsymbol{b}_{noisy} \odot \epsilon^b + (\mathbf{W}_{noisy} \odot \epsilon^w)\boldsymbol{x}), \tag{4}$$

where ϵ^b and ϵ^w are random variables, and $\odot$ denotes the element-wise product. This transformation can then be used in place of the standard linear $\boldsymbol{y} = \boldsymbol{b} + \mathbf{W}\boldsymbol{x}$. Over time, the network can learn to ignore the noisy stream, but will do so at different rates in different parts of the state space, allowing state-conditional exploration with a form of self-annealing.

The Integrated Agent

In this paper we integrate all the aforementioned components into a single integrated agent, which we call *Rainbow*.

First, we replace the 1-step distributional loss (3) with a multi-step variant. We construct the target distribution by contracting the value distribution in S_{t+n} according to the cumulative discount, and shifting it by the truncated n-step discounted return. This corresponds to defining the target distribution as $d_t^{(n)} = (R_t^{(n)} + \gamma_t^{(n)}\boldsymbol{z},\ \ \boldsymbol{p}_{\overline{\theta}}(S_{t+n}, a^*_{t+n}))$. The resulting loss is

$$D_{\text{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t)\,,$$

where, again, $\Phi_{\boldsymbol{z}}$ is the projection onto $\boldsymbol{z}$.

We combine the multi-step distributional loss with double Q-learning by using the greedy action in S_{t+n} selected according to the *online network* as the bootstrap action a^*_{t+n}, and evaluating such action using the *target network*.

In standard proportional prioritized replay (Schaul et al. 2015) the absolute TD error is used to prioritize the transitions. This can be computed in the distributional setting, using the mean action values. However, in our experiments all distributional Rainbow variants prioritize transitions by the KL loss, since this is what the algorithm is minimizing:

$$p_t \propto \left(D_{\text{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t)\right)^\omega\,.$$

The KL loss as priority might be more robust to noisy stochastic environments because the loss can continue to decrease even when the returns are not deterministic.

The network architecture is a dueling network architecture adapted for use with return distributions. The network

has a shared representation $f_\xi(s)$, which is then fed into a value stream v_η with N_{atoms} outputs, and into an advantage stream a_ξ with $N_{\text{atoms}} \times N_{\text{actions}}$ outputs, where $a^i_\xi(f_\xi(s), a)$ will denote the output corresponding to atom i and action a. For each atom z^i, the value and advantage streams are aggregated, as in dueling DQN, and then passed through a softmax layer to obtain the normalised parametric distributions used to estimate the returns' distributions:

$$p^i_\theta(s,a) = \frac{\exp(v^i_\eta(\phi) + a^i_\psi(\phi,a) - \overline{a}^i_\psi(s))}{\sum_j \exp(v^j_\eta(\phi) + a^j_\psi(\phi,a) - \overline{a}^j_\psi(s))},$$

where $\phi = f_\xi(s)$ and $\overline{a}^i_\psi(s) = \frac{1}{N_{\text{actions}}} \sum_{a'} a^i_\psi(\phi, a')$.

We then replace all linear layers with their noisy equivalent described in Equation (4). Within these noisy linear layers we use factorised Gaussian noise (Fortunato et al. 2017) to reduce the number of independent noise variables.

Experimental Methods

We now describe the methods and setup used for configuring and evaluating the learning agents.

Evaluation Methodology. We evaluated all agents on 57 Atari 2600 games from the arcade learning environment (Bellemare et al. 2013). We follow the training and evaluation procedures of Mnih et al. (2015) and van Hasselt et al. (2016). The average scores of the agent are evaluated during training, every 1M steps in the environment, by suspending learning and evaluating the latest agent for 500K frames. Episodes are truncated at 108K frames (or 30 minutes of simulated play), as in van Hasselt et al. (2016).

Agents' scores are normalized, per game, so that 0% corresponds to a random agent and 100% to the average score of a human expert. Normalized scores can be aggregated across all Atari levels to compare the performance of different agents. It is common to track the *median* human normalized performance across all games. We also consider the number of games where the agent's performance is above some fraction of human performance, to disentangle where improvements in the median come from. The *mean* human normalized performance is potentially less informative, as it is dominated by a few games (e.g., Atlantis) where agents achieve scores orders of magnitude higher than humans do.

Besides tracking the median performance as a function of environment steps, at the end of training we re-evaluate the best agent snapshot using two different testing regimes. In the *no-ops starts* regime, we insert a random number (up to 30) of no-op actions at the beginning of each episode (as we do also in training). In the *human starts* regime, episodes are initialized with points randomly sampled from the initial portion of human expert trajectories (Nair et al. 2015); the difference between the two regimes indicates the extent to which the agent has over-fit to its own trajectories.

Hyper-parameter tuning. All Rainbow's components have a number of hyper-parameters. The combinatorial space of hyper-parameters is too large for an exhaustive search, therefore we have performed limited tuning. For each component, we started with the values used in the paper that introduced this component, and tuned the most sensitive among hyper-parameters by manual coordinate descent.

DQN and its variants do not perform learning updates during the first $200K$ frames, to ensure sufficiently uncorrelated updates. We have found that, with prioritized replay, it is possible to start learning sooner, after only $80K$ frames.

DQN starts with an exploration ϵ of 1, corresponding to acting uniformly at random; it anneals the amount of exploration over the first 4M frames, to a final value of 0.1 (lowered to 0.01 in later variants). Whenever using Noisy Nets, we acted fully greedily ($\epsilon = 0$), with a value of 0.5 for the σ_0 hyper-parameter used to initialize the weights in the noisy stream[1]. For agents without Noisy Nets, we used ϵ-greedy but decreased the exploration rate faster than was previously used, annealing ϵ to 0.01 in the first $250K$ frames.

We used the Adam optimizer (Kingma and Ba 2014), which we found less sensitive to the choice of the learning rate than RMSProp. DQN uses a learning rate of $\alpha = 0.00025$ In all Rainbow's variants we used a learning rate of $\alpha/4$, selected among $\{\alpha/2, \alpha/4, \alpha/6\}$, and a value of 1.5×10^{-4} for Adam's ϵ hyper-parameter.

The value of n in multi-step learning is a sensitive hyper-parameter of Rainbow. We compared values of $n = 1,\ 3,$ and 5. We observed that both $n = 3$ and 5 did well initially, but overall $n = 3$ performed the best by the end. For replay prioritization we used the recommended proportional variant, with priority exponent ω of 0.5, and linearly increased the importance sampling exponent β from 0.4 to 1 over the course of training. The priority exponent ω was tuned comparing values of $\{0.4, 0.5, 0.7\}$. Using the KL loss of distributional DQN as priority, we have observed that performance is very robust to the choice of ω.

The hyper-parameters (see Table 1) are identical across all 57 games, i.e., the Rainbow agent really is a *single* agent setup that performs well across all the games.

[1]The noise was generated on the GPU. Tensorflow noise generation can be unreliable on GPU. If generating the noise on the CPU, lowering σ_0 to 0.1 may be helpful.

Parameter	Value
Min history to start learning	80K frames
Adam learning rate	0.0000625
Exploration ϵ	0.0
Noisy Nets σ_0	0.5
Target Network Period	32K frames
Adam ϵ	1.5×10^{-4}
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Multi-step returns n	3
Distributional atoms	51
Distributional min/max values	$[-10, 10]$

Table 1: Rainbow hyper-parameters

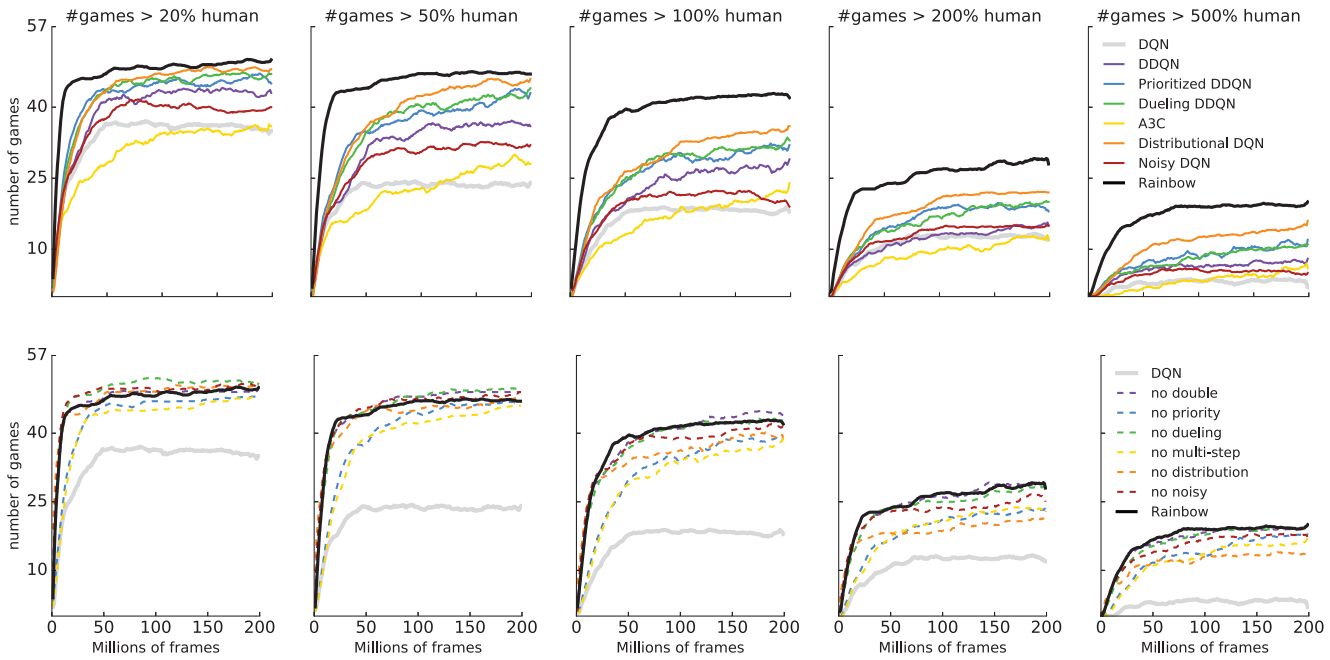

Figure 2: Each plot shows, for several agents, the number of games where they have achieved at least a given fraction of human performance, as a function of time. From left to right we consider the 20%, 50%, 100%, 200% and 500% thresholds. On the first row we compare Rainbow to the baselines. On the second row we compare Rainbow to its ablations.

Analysis

In this section we analyse the main experimental results. First, we show that Rainbow compares favorably to several published agents. Then we perform ablation studies, comparing several variants of the agent, each corresponding to removing a single component from Rainbow.

Comparison to published baselines. In Figure 1 we compare the Rainbow's performance (measured in terms of the median human normalized score across games) to the corresponding curves for A3C, DQN, DDQN, Prioritized DDQN, Dueling DDQN, Distributional DQN, and Noisy DQN. We thank the authors of the Dueling and Prioritized agents for providing the learning curves of these, and report our own re-runs for DQN, A3C, DDQN, Distributional DQN and Noisy DQN. The performance of Rainbow is better than any of the baselines by a large margin, both in data efficiency, as well as in final performance. Note that we match final performance of DQN after 7M frames, surpass the best final performance of these baselines in 44M frames, and reach substantially improved final performance.

In the final evaluations of the agent, after the end of training, Rainbow achieves a median score of 231% in the no-ops regime; in the human starts regime we measured a median score of 153%. In Table 2 we compare these scores to the published median scores of the individual baselines.

In Figure 2 (top row) we plot the number of games where an agent has reached some specified level of human normalized performance. From left to right, the subplots show on how many games the different agents have achieved at least 20%, 50%, 100%, 200% and 500% human normalized performance. This allows us to identify where the overall improvements in performance come from. Note that the gap in performance between Rainbow and other agents is apparent at all levels of performance: the Rainbow agent is improving scores on games where the baseline agents were already good, as well as improving in games where baseline agents are still far from human performance.

Agent	no-ops	human starts
DQN	79%	68%
DDQN (*)	117%	110%
Prioritized DDQN (*)	140%	128%
Dueling DDQN (*)	151%	117%
A3C (*)	-	116%
Noisy DQN	118%	102%
Distributional DQN	185%	125%
Rainbow	231%	153%

Table 2: Median normalized scores of the best agent snapshots for Rainbow and baselines. For methods marked with an asterisk, the scores come from the corresponding publication. DQN's scores comes from the dueling networks paper, since DQN's paper did not report scores for all 57 games. The others scores come from our own implementations.

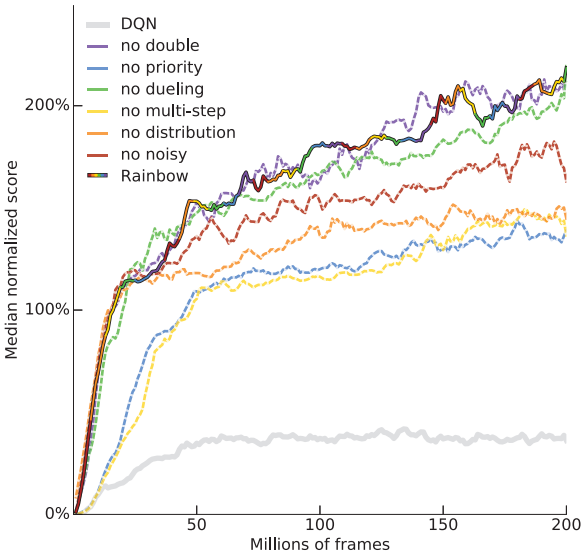

Figure 3: Median human-normalized performance across 57 Atari games, as a function of time. We compare our integrated agent (rainbow-colored) to DQN (gray) and to six different ablations (dashed lines). Curves are smoothed with a moving average over 10 points.

Learning speed. As in the original DQN setup, we ran each agent on a single GPU. The 7M frames required to match DQN's final performance correspond to less than 10 hours of wall-clock time. A full run of 200M frames corresponds to approximately 10 days, and this varies by less than 20% between all of the discussed variants. The literature contains many alternative training setups that improve performance as a function of wall-clock time by exploiting parallelism, e.g., Nair et al. (2015), Salimans et al. (2017), and Mnih et al. (2016). Properly relating the performance across such very different hardware/compute resources is non-trivial, so we focused exclusively on algorithmic variations, allowing apples-to-apples comparisons. While we consider them to be important and complementary, we leave questions of scalability and parallelism to future work.

Ablation studies. Since Rainbow integrates several different ideas into a single agent, we conducted additional experiments to understand the contribution of the various components, in the context of this specific combination.

First, we performed an ablation study. In each ablation, we removed a single component from the full Rainbow combination, and trained the resulting agent on all Atari games. Figure 3 compares median normalized scores of Rainbow to the six ablated variants. Figure 2 (bottom row) shows a more detailed breakdown of how these ablations perform relative to different thresholds of human normalized performance, and Figure 4 shows the gain or loss from each ablation for every game, averaged over the full learning run.

Prioritized replay and multi-step learning were the two most crucial components of Rainbow, in that removing either component caused a large drop in median performance. Unsurprisingly, the removal of either of these hurt early performance. Perhaps more surprisingly, the removal of multi-step learning also hurt final performance. Zooming in on individual games (Figure 4), we see both components helped almost uniformly across games (Rainbow performed better than either ablation in 53 games out of 57).

Distributional Q-learning ranked immediately below the previous techniques for relevance to the agent's performance. Notably, in early learning no difference is apparent, as shown in Figure 3, where for the first 40 million frames the distributional-ablation performed as well as the full agent. However, without distributions, the performance of the agent then started lagging behind. When the results are separated relatively to human performance in Figure 2, we see that the distributional-ablation primarily seems to lags on games that are above human level or near it.

In terms of median performance, the agent performed better when Noisy Nets were included; when these are removed and exploration is delegated to the traditional ϵ-greedy mechanism, performance was worse in aggregate (red line in Figure 3). While the removal of Noisy Nets produced a large drop in performance for several games, it also provided small increases in other games (Figure 4).

In aggregate, we did not observe a significant difference when removing the dueling network from the full Rainbow. The median score, however, hides the fact that the impact of Dueling differed between games, as shown by Figure 4. Figure 2 shows that Dueling perhaps provided some improvement on games with above-human performance levels (# games $> 200\%$), and some degradation on games with sub-human performance (# games $> 20\%$).

Also in the case of double Q-learning, the observed difference in median performance (Figure 3) is limited, with the component sometimes harming or helping depending on the game (Figure 4). To further investigate the role of double Q-learning, we compared the predictions of our trained agents to the actual discounted returns computed from clipped rewards. Comparing Rainbow to the agent where double Q-learning was ablated, we observed that the actual returns are often higher than 10 and therefore fall outside the support of the distribution, spanning from -10 to $+10$. This leads to underestimated returns, rather than overestimations. We hypothesize that clipping the values to this constrained range counteracts the overestimation bias of Q-learning. Note, however, that the importance of double Q-learning may increase if the support of the distributions is expanded.

Discussion

We have demonstrated that several improvements to DQN can be successfully integrated into a single learning algorithm that achieves state-of-the-art performance. Moreover, we have shown that within the integrated algorithm, all but one of the components provided clear performance benefits. There are many more algorithmic components that we were not able to include, which would be promising candi-

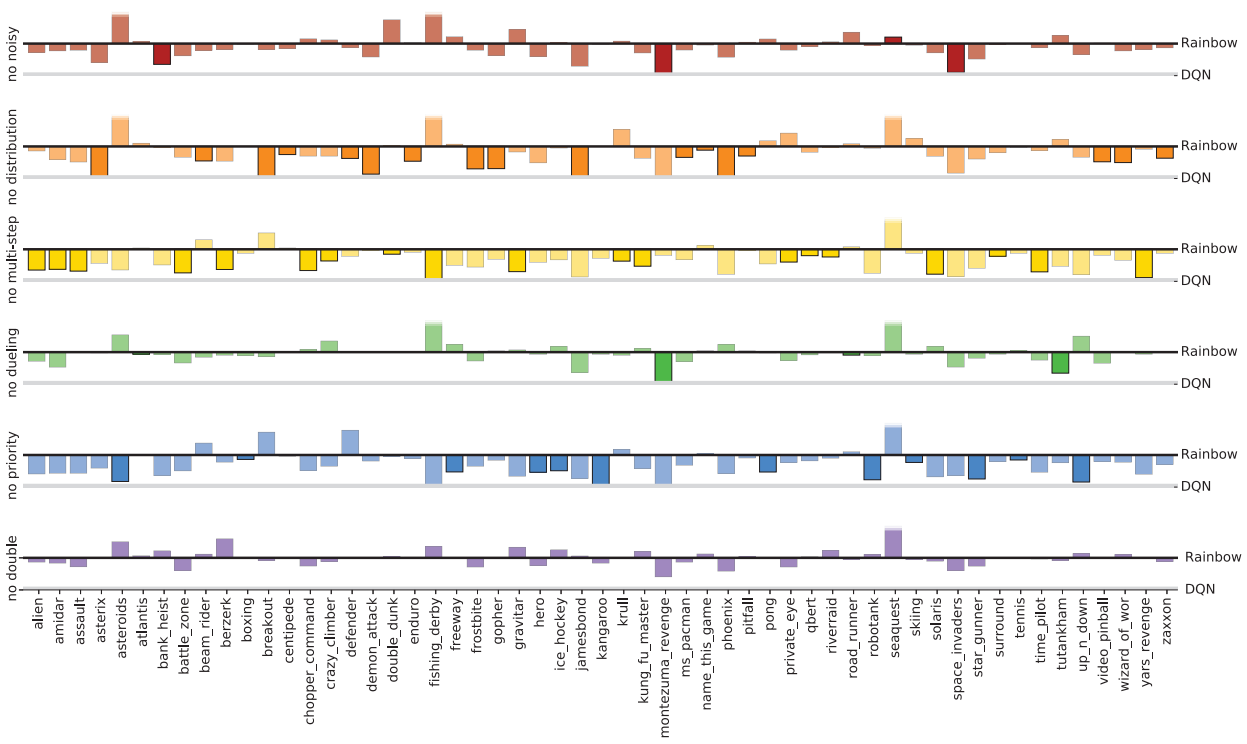

Figure 4: Performance drops of ablation agents: Performance is area under the curve, normalized relative to Rainbow and DQN. Venture and Bowling, where DQN outperformed Rainbow, are omitted. Ablations leading to the largest drop are highlighted per game. Prioritization and multi-step help almost uniformly across games, other components' impact differs per game.

dates for further experiments on integrated agents. Among the many possible candidates, we discuss several below.

We focused on value-based methods in the Q-learning family, but similar ideas may benefit also policy-based RL algorithms such as TRPO (Schulman et al. 2015), or actor-critic methods (Mnih et al. 2016; O'Donoghue et al. 2016).

A number of algorithms exploit sequences of data to improve learning efficiency. Optimality tightening (He et al. 2016) uses multi-step returns to construct additional inequality bounds, instead of just replacing the 1-step targets in Q-learning. Eligibility traces allow a soft combination over n-step returns (Sutton 1988). However, sequential methods all leverage more computation per update than the multi-step targets used in Rainbow. Also, the combination of prioritized replay with sequence data is still an open problem.

Episodic control (Blundell et al. 2016) also focuses on data efficiency, and was shown to be very effective in some domains. It improves early learning by using episodic memory as a complementary learning system, capable of immediately re-enacting successful action sequences.

Besides Noisy Nets, many exploration methods have been proposed: e.g. Bootstrapped DQN (Osband et al. 2016), intrinsic motivation (Stadie, Levine, and Abbeel 2015) and count-based exploration (Bellemare et al. 2016). Combining these with Rainbow is fruitful subject for further research.

We focused on the core learning updates, without exploring alternative computational architectures. Asynchronous learning from parallel copies of the environment, as in A3C (Mnih et al. 2016), Gorila (Nair et al. 2015), or Evolution Strategies (Salimans et al. 2017), can speed up learning in wall-clock time, although at the cost of data efficiency.

Hierarchical RL has also been applied with success to several complex Atari games. Among successful applications of HRL we highlight h-DQN (Kulkarni et al. 2016a) and Feudal Networks (Vezhnevets et al. 2017).

The state representation could be improved through the use of auxiliary tasks such as pixel or feature control (Jaderberg et al. 2016), supervised predictions (Dosovitskiy and Koltun 2016) or successor features (Kulkarni et al. 2016b).

To evaluate Rainbow fairly against baselines, we followed the common domain modifications of frame-stacking, reward clipping, and fixed action-repetition. These may be replaced by more principled techniques. Recurrent networks (Hausknecht and Stone 2015) can learn temporal state representations, replacing frame-stacking. Pop-Art (van Hasselt et al. 2016) enables learning from raw rewards. Fine-grained action repetition (Sharma, Lakshminarayanan, and Ravindran 2017) learns the number of action repetitions. In general, we believe that exposing the real game to agents is a promising direction for future research.

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation plat-

form for general agents. *J. Artif. Intell. Res. (JAIR)* 47:253–279.

Bellemare, M. G.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. In *NIPS*.

Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. A distributional perspective on reinforcement learning. In *ICML*.

Blundell, C.; Uria, B.; Pritzel, A.; Li, Y.; Ruderman, A.; Leibo, J. Z.; Rae, J.; Wierstra, D.; and Hassabis, D. 2016. Model-Free Episodic Control. *ArXiv e-prints*.

Dosovitskiy, A., and Koltun, V. 2016. Learning to act by predicting the future. *CoRR* abs/1611.01779.

Fortunato, M.; Azar, M. G.; Piot, B.; Menick, J.; Osband, I.; Graves, A.; Mnih, V.; Munos, R.; Hassabis, D.; Pietquin, O.; Blundell, C.; and Legg, S. 2017. Noisy networks for exploration. *CoRR* abs/1706.10295.

Hausknecht, M., and Stone, P. 2015. Deep recurrent Q-learning for partially observable MDPs. *arXiv preprint arXiv:1507.06527*.

He, F. S.; Liu, Y.; Schwing, A. G.; and Peng, J. 2016. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *CoRR* abs/1611.01606.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2016. Reinforcement learning with unsupervised auxiliary tasks. *CoRR* abs/1611.05397.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. B. 2016a. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR* abs/1604.06057.

Kulkarni, T. D.; Saeedi, A.; Gautam, S.; and Gershman, S. J. 2016b. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*.

Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8(3):293–321.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing atari with deep reinforcement learning. *CoRR* abs/1312.5602.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*.

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

O'Donoghue, B.; Munos, R.; Kavukcuoglu, K.; and Mnih, V. 2016. Pgq: Combining policy gradient and q-learning. *CoRR* abs/1611.01626.

Osband, I.; Blundell, C.; Pritzel, A.; and Roy, B. V. 2016. Deep exploration via bootstrapped dqn. In *NIPS*.

Salimans, T.; Ho, J.; Chen, X.; and Sutskever, I. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR* abs/1703.03864.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. In *Proc. of ICLR*.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.; and Abbeel, P. 2015. Trust region policy optimization. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, 1889–1897. JMLR.org.

Sharma, S.; Lakshminarayanan, A. S.; and Ravindran, B. 2017. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*.

Stadie, B. C.; Levine, S.; and Abbeel, P. 2015. Incentivizing exploration in reinforcement learning with deep predictive models. *CoRR* abs/1507.00814.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA.

Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3(1):9–44.

Tieleman, T., and Hinton, G. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2):26–31.

van Hasselt, H.; Guez, A.; Guez, A.; Hessel, M.; Mnih, V.; and Silver, D. 2016. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems 29*, 4287–4295.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double Q-learning. In *Proc. of AAAI*, 2094–2100.

van Hasselt, H. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems 23*, 2613–2621.

Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. Feudal networks for hierarchical reinforcement learning. *CoRR* abs/1703.01161.

Wang, Z.; Schaul, T.; Hessel, M.; van Hasselt, H.; Lanctot, M.; and de Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, 1995–2003.