Investigating Sample Inefficiency in Reinforcement Learning

Adrian Coutsoftides, Doddi El-Gabry, Vytenis Šliogeris and Alex Vavvas

University of Surrey

Introduction

While reinforcement learning alludes to promises nearing Artificial General Intelligence, the purpose of this project is motivated by the very criticisms that refute the efficacy of RL [Irpan, 2018] Such claims point to RL's sample inefficiency, complex reward function design, generalization problem and arguments for other, more classical robotics techniques. We investigate the validity of these claims as well as architectures published throughout by measuring the performance of variations to the original Deep Q-Network introduced by [Mnih et al., 2015] on the OpenAI Gym™ Atari 2600 suite. We measure the sample efficiency of three different models on a selection of games in addition to introducing a novel variation to the original Rainbow-DQN algorithm. Our results indicate that sample efficiency is closely related agent's ability to represent distributions of possible outcomes rather than expected returns.

Literature Review

Deep reinforcement learning combines deep neural networks and optimal control theory to empower learning agents to take optimal actions in complex environments that can be either fully or partially observed. An optimal control problem consists of the following:

- An environment formed of a set of states $s \in \{\mathcal{S}\}$ and terminal state s_τ
- A set of possible actions $a \in \{\mathcal{A}\}$ that transition the environment $s \to s'$
- A scalar reward $r \in \{\mathbb{R}\}$ derived from transitions into favourable states $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$

The agent's reward structure is dependent on the goal of optimization and is usually crafted by expert knowledge. In order to achieve this goal, the agent must maximize the scalar reward it acquires over τ time steps. As the agent can only ever estimate future rewards they are *discounted* by a factor of $0 < \gamma < 1$ as shown in equation (1).

Q Learning

$$\begin{aligned} r_1 + r_2 + \ldots + r_\tau &= \sum_\tau r_t \\ \Rightarrow r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots + \gamma^{\tau-1} r_\tau &= \sum_\tau \gamma^{t-1} r_t \\ \Rightarrow \sum_\tau \gamma^{t-1} r_t = G_t &= r_1 + \gamma G_{t+1} \end{aligned} \tag{1}$$

$$Q(s,a) \leftarrow Q(s_t, a_t) + \alpha[(r_t + \gamma \max_{a'} Q'(s_{t+1}, a')) - Q(s_t, a_t)] \tag{2}$$

Where γ is denoted as the discount factor, indicating that agent should value more immediate reward rather than the later rewards, as those can be more reliably predicted. In **episodic** settings, the agent interacts with the environment until an absorbing state is reached, which represents a terminal state or a set of states cyclically connected from which the agent cannot transition out of. At the beginning of each episode the environment is reset to the starting state s_0 and the training process is repeated.
In order to select actions to take in the environment the agent considers the value of a particular state s_t to be the **expected sum of cumulative discounted rewards** if the agent started in s_t and acted optimally until s_τ. The optimal action is that which maximises the expected discounted returns within each state. To progress in the environment, the agent must form a policy $\pi(a \mid s)$ that maps states to actions. The value of a particular action in a particular state is given by $Q(s, a)$, the agent updates its estimates in an online fashion by bootstrapping from previous estimates with some learning rate $0 < \alpha < 1$.
Equation (2) describes the Q learning procedure, which is considered an *off-policy* algorithm. This is because Q' and Q represent two different policies $\pi' \neq \pi$. In practice implemented by two different *Convolutional Neural Networks* (CNNs) that take the pixel frames of the game as input and output a vector of action values $\mathbf{a}$, one for each action. Q' is denoted the *target policy* and Q is denoted the *behaviour policy*, this is the policy that generates the data by taking actions in the environment. By improving the behaviour policy through learning in the environment, and then making the target policy greedy w.r.t the improved behaviour policy ($\max_{a'} Q'(s_{t+1}, a')$), we monotonically improve our estimates of the action values. Assuming that $\pi'(a \mid s) \geq \pi(a \mid s)$ if and only if $Q'(s,a) > Q(s,a)$, *generalised policy iteration* [Sutton, 2018] guarantees convergence to an optimal policy $\boldsymbol{\pi}^*$.

Deep Q Learning

$$\delta_t = r + \gamma Q(s, a; \boldsymbol{\theta}^-) \tag{3}$$

$$\begin{aligned} L(\delta_t, Q(s,a;\boldsymbol{\theta})) &= \|\delta_t - Q(s,a;\boldsymbol{\theta})\|_2 \\ \frac{\partial L(\delta_t, Q(s,a;\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} &= \frac{\partial(\delta_t - Q(s,a;\boldsymbol{\theta}))^2}{\partial \boldsymbol{\theta}} \\ &= (\delta_t - Q(s,a;\boldsymbol{\theta})) \cdot \frac{\partial(\delta_t - Q(s,a;\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \\ &= (\delta_t - Q(s,a;\boldsymbol{\theta}))\nabla_{\boldsymbol{\theta}} Q(s,a;\boldsymbol{\theta}) \end{aligned} \tag{4}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} L(\delta_t, Q(s,a;\boldsymbol{\theta})) \tag{5}$$

Q learning was effectively extended to incorporate deep learning methods by contextualising the Q learning update as a supervised learning update with eq. 4 acting as the target calculated from the target network's parameters $\boldsymbol{\theta}^-$. The target network's parameters $\boldsymbol{\theta}^-$ do not get updated with each training step. Instead, every k steps the parameters of $\boldsymbol{\theta}$ get copied over to $\boldsymbol{\theta}^-$, making k another trainable parameter. The behaviour network seeks to minimize the L_2 loss between its predicted action values and the values derived from the target network; and so eq. 4 derives the gradient calculation for the weight update of the behaviour network which is applied as 5 for some learning rate $0 < \eta < 1$. Critically, [Mnih et al., 2015] make use of a experience buffer to store transitions in the form of a tuple (s, a, s', r). The replay buffer is uniformly sampled for a `batch_size` number of transitions. The Q values of each action are estimated according to eq. 2 and then weights are updated with eq. 5.

Rainbow DQN [Hessel et al., 2017a] is an effort to combine all previous improvements to the original DQN into a single model. Specifically, rainbow combines six of these improvements, each one tackling a single limitation of the original DQN. These can be seen in the list below.

- Double Q-Learning - Reduces over-estimations that arise from 2
- Prioritized Replay - Prioritizes more recent experiences when sampling the buffer.
- Dueling Networks - Two separate streams, value and advantage, merged into a single.
- Multi-step Learning - Aims to make rewards more representative by replacing the rewards with rewards after some number of steps, as shown in 6.
- Noisy Nets - Replaces ϵ-greedy exploration by introducing a noisy linear layer that has different effects at different parts of the state space.
- Distributional RL - Instead of approximating the expected return with approximate a distribution of returns.

Rainbow was able to achieve SOTA performance, outperforming all previous approaches.

Multi-step learning

$$R_t^n \equiv \sum_{k=0}^{n-1} R_{t+k+1} \tag{6}$$

Given some number of steps, n, rewards at timestep t in the memory tuples are replaced by R_t^n, which is just the sum of the rewards from time step $t+1$ to $t+n$. Therefore each value in a memory tuple better represents the future values that one can get from a particular state better, helping with the sparseness of data.

Noisy Nets

$$y = (b + Wx) + (b_{noisy} \odot \epsilon^b + (W_{noisy} \odot \epsilon^W)x) \tag{7}$$

Noisy nets [Fortunato et al., 2017] are a substitute for the epsilon-greedy exploration policy. The linear layers in neural networks get replaced by ones represented in eq. 7. The first half of the equation, $y = (b + Wx)$, is the classical linear layer equation, where x is the input vector, y is the output vector and W and b are the parameters that get trained using linear regression.

The second half of the equation is similar to the classical linear layer, except the trainable parameters b_{noisy} and W_{noisy} are element-wise multiplied by ϵ^b and ϵ^W respectively. ϵ is randomly sampled Gaussian noise. At the start of the learning process the noise serves to randomise the action selection process, in essence introducing exploration. As the network learns, it decreases the magnitude of b_{noisy} and W_{noisy}, making it self-anneal.
Using a single estimator for the loss function has been show to over-estimate the rewards [Hasselt, 2010]. Their proposed solution was to change the loss function to use two estimators instead, as shown in eq. 11.

Prioritised Experience Replay

$$\begin{aligned} p_i &= |\delta_i| + \epsilon \\ P(i) &= \left(\frac{p_i}{\sum_k p_k}\right)^\alpha \end{aligned} \tag{8}$$

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{9}$$

$$\nabla_\theta E = w_j \delta_j \nabla_\theta Q(S_{j-1}, A_{j-1}\ ;\ \theta) \tag{10}$$

The motivation behind *Prioritised Experience Replay* [Hessel et al., 2017b] is the notion that given a batch of pre-recorded transitions, the agent should ideally prioritise learning from memories with more valuable learning experiences over others, rather than randomly sampling from the replay buffer.The magnitude of the δ_t error can be interpreted as the amount of "surprise" associated with the that sequence, if our estimated Q value for a state action pair was far from our target, the the result of our actions will always deviate from expectations because we did not form a reliable way to ascertain the outcome of events. Updating our estimates for events in which we were surprised will improve the agent's ability to predict future states. Eq. 8 describes the procedure of assign a priority to a memory, where ϵ is some small positive constant to prevent the priority p_i from being zero and $P(i)$ is the probability of the transition being chosen for replay. The exponent α determines the amount of prioritisation used where $\alpha = 0$ is uniform (regular experience replay). However the use of such a distribution introduces bias into the updates of the Q value. The estimation of an expected value with stochastic updates relies on those updates coming from the same distribution as our estimation, by constructing the distribution as in (2.51), we are now drawing on observations from a different distribution than the one underlying the MDP. This is corrected for with importance sampling weights as in eq. 9.

Double Q Learning

$$\delta_i = R_{t+1} + \gamma Q(S_{t+1}, \arg\max\ Q(S_{t+1}, A_{t+1}\ ;\ \boldsymbol{\theta})\ ;\ \boldsymbol{\theta}^-) \tag{11}$$

In eq. 2 we select actions according to $\underset{a}{argmax}\, Q(s, a; \boldsymbol{\theta}^-)$, this incurs a significant *positive* bias due to the nature of the maximization. The error occurs because the same samples are used to determine both the value of each action and to select the maximising action. [Hasselt, 2010] address this within the context of deep Q learning by proposing a variant of the Bellman error δ_t as seen in 11. The main difference is that when evaluating the next state using the target network, instead of choosing the best action from the target network's perspective, the best action from the behaviour network's perspective is chosen. This corrects for the over-estimation and speeds up the training process.

Dueling Networks

$$Q(s,a) = V(s) + A(s,a) \tag{12}$$

$$\begin{aligned} A(s,a) = 0 &\iff V^*(s) = \max_a Q^*(s,a) \\ \to A(s,a) &= Q^*(s,a) - V^*(s) = 0 \end{aligned} \tag{13}$$

[Wang et al., 2015] observed that the action value function can be decomposed further into a value and advantage function. As the value of a state is the value of the most optimal action, the relative advantage of the action to itself is 0 (eq. 13).

Investigating Sample Inefficiency in Reinforcement Learning

Adrian Coutsoftides, Doddi El-Gabry, Vytenis Šliogeris and Alex Vavvas

University of Surrey

Given only a Q function we cannot recover the value and advantage and value uniquely due to the full-connected nature of the hidden layers of the neural network. Changes in the weights affect the entire space of approximation. Instead the authors decompose the output layer of the DQN into two seperate heads, one that calculates the advantage and another to calculate the value (as a scalar). This allows the network to further discriminate and capture small fluctuations in the advantage of a particular action while isolating the weights that calculate the value of a state, making the model more robust to parameter updates. Where the advantage is a measure of the advantage of taking an action in that state relative to other possible actions.

Distributional Bellman Operator

$$\begin{gathered} x \in \mathbf{S}, \quad x' \coloneqq x_{t+1} \\ \mathcal{T}^{\pi} Q(x,a) = \mathbb{E}[R(x,a)] + \gamma \mathop{\mathbb{E}}_{P,\pi}[Q(x',a')] \\ \Rightarrow \mathcal{T} Q(x,a) = \mathbb{E}[R(x,a)] + \gamma \mathop{\mathbb{E}}_{P,\pi}[\max_{a' \in A} Q(x',a')] \end{gathered} \tag{14}$$

It can be shown that Q learning converges to a fixed point represented by a deterministic optimal policy $\pi^* \approx \pi_\theta$ [Sutton, 2018]. This optimal policy is approximated by the neural network's parameters arbitrarily well, however, this approximation is only in their expectations. Critically, two distributions can share the same mean but have a different "shape", hence modelling expectations fails to capture properties such as multi-modalities of the underlying distribution of the Markov decision process.
Motivated by these shortcomings, [Bellemare et al., 2017] proposed an alternative distributional perspective on reinforcement learning. The authors reinterpret eq. 2 under a distributional variant; first they contextualize the Q learning update under the *Bellman operator* $\mathcal{T}^{\pi}$ and in the control setting as the application of the *"optimality operator"* $\mathcal{T}$: This reflects the same objective as 2, where successive applications of the Bellman operator produce new policies parameterised by the updated estimates to the Q value function.

Three sources of randomness arise under this interpretation:

- Randomness in reward $R(x,a)$.
- Randomness in the transition $\mathcal{P}^{\pi}$.
- Randomness in the next state-value distribution $Z(x',a')$.

These sources are important to observe when considering if the distributional variant of the Bellman equation converges to a fixed point Z^{π}. In order to measure the relative distance between an arbitrary distribution, the authors utilise the *Wasserstein* distance defined on cumulative density functions (c.d.f) of arbitrary probability density functions (p.d.f). If a random variable is distributed according to $F(x)$ then its c.d.f, also known as a quantile function, is $F^{-1}(x)$.These sources are important to observe when considering if the distributional variant of the Bellman equation converges to a fixed point Z^{π}. The *Wasserstein* distance (eq. 15) is an L_p metric for some $p \in \mathbb{N}$ that measures the distance between two random variables U, V on the space of possible samples $\omega \in \Omega$. This distance represents the smallest possible difference between the probability density of each sample whose support is the metric space $[0,1]$.

The Wasserstein Metric

$$\begin{aligned} W_p(F,G) &= \left(\int_0^1 |F_U^{-1}(\omega) - F_V^{-1}(\omega)|^p \, \mathrm{d}x \right)^{\frac{1}{p}} \\ &= \inf_{U,V} \|U - V\|_p \end{aligned} \tag{15}$$

$$\bar{d}_p(Z_1, Z_2) \coloneqq \sup_{x,a} W_p(Z_1(x,a), Z_2(x,a)) \tag{16}$$

To measure the distance between value distributions Z_1, Z_2, they define the maximal form of the Wasserstein metric as the highest element-wise absolute difference between the two cumulative density functions as in eq. 16.

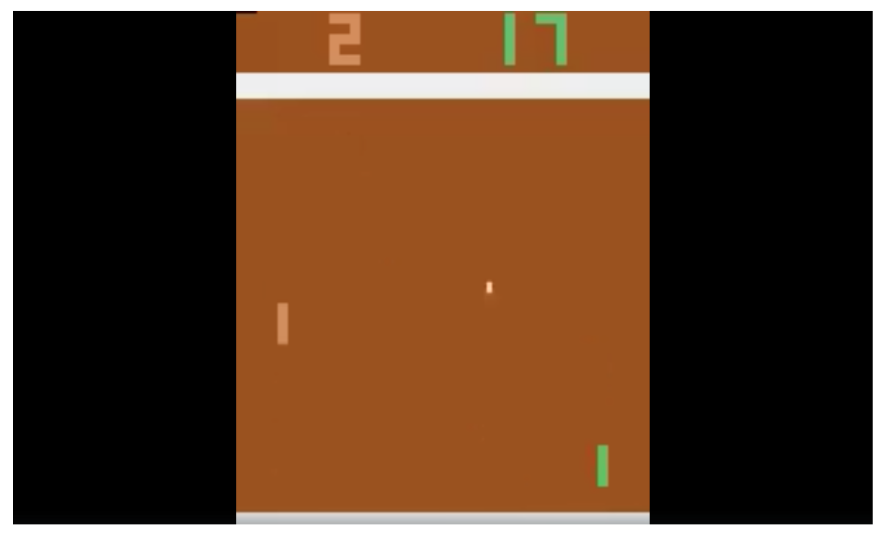
Although [Bellemare et al., 2017] develop strong convergence guarantees for the $\bar{d}_p$ metric, it is difficult to apply the metric directly as gradient descent gradually changes the weights over the course of training. This means that the support of the distribution drifts over t timesteps, resulting in disjoint supports when measured under $\bar{d}_p$ such that $\bar{d}_p(Z_t, Z_{t'}) \to \infty$ where $t \neq t'$. We discuss the impact of this factor in **Challenges** and **Implementation**.

Methods

In order to investigate the claims of [Irpan, 2018], we needed to measure the efficacy of different models, each one built on the results of the last, in order to investigate what components were the most effective in increasing sample efficiency. Being additionally limited by the hardware at hand, we also had to carefully select components so that we could train the agents in a reasonable amount of time and still observe any performance gains. As we are measuring relative performance gains, even incremental improvements to performance will prove useful in analysis.

As discussed in **Implementation**, we test three models on a variety of tasks with varying levels of difficulty. Given the results of Dabney et al.[Dabney et al., 2017], we assume that we will see the largest performance benefit from the introduction of quantile regression learning, due to added expressivity of the representation. Although we expect performance to positively correlate with each added improvement, we are interested specifically in the relative difference in average reward for each timestep between the models. We observe an increase in performance over the training compared across all three games on the two models.
In addition to limited processing power, time restrictions were also a factor in our experimental design. Many experiments such as [Hessel et al., 2017b] and [Mnih et al., 2015] usually implement learning rates as low as 6.25×10^{-5} as this facilitates a more stable training phase, although it also significantly increases the time of the training phase. With this in mind, we avoided selecting traditionally difficult environments such as Montezuma's Revenge [Aytar et al., 2018] and instead selected environments with simple actions and rewards.
Rather than spend additional time trying to implement the C51 algorithm, we found their follow-up work much more intuitive, additionally we sought to maximize the performance of our algorithms over such few time steps, so integrating the subsequent work was a natural and necessary extension of our experiment.

(a) A subfigure



(b) A subfigure

Figure: A figure with two subfigures

(a) A subfigure

Figure: A figure with two subfigures

Challenges

The classic challenges of agents learning to play Atari games is that the state-space of the game is usually incredibly vast. For example, in Pong, the number of possible states is each possible position for the player multiplied by each possible position of the opponent multiplied by each possible position of the ball. Therefore Reinforcement Learning seemed like a good model for such a problem as, instead of needing a dataset of game-frames, which would be problematic to construct, it can learn to evaluate a state and then derive an optimal policy using the learnt evaluation function. However, reinforcement learning poses challenges of its own.

Firstly, the rewards in pong are very sparse relatively to the number of game frames played, since a reward is only achieved when either player scores. This means that during the exploration stage the agent must get lucky to score so that it could associate an action with a positive reward.

Secondly, the agent receives pixel values of the monitor instead of the actual coordinates for each of the objects. This forced us to use a CNN, however, the agent has to at least learn to identify itself and the ball so that it could take meaningful actions to move in front of the ball.

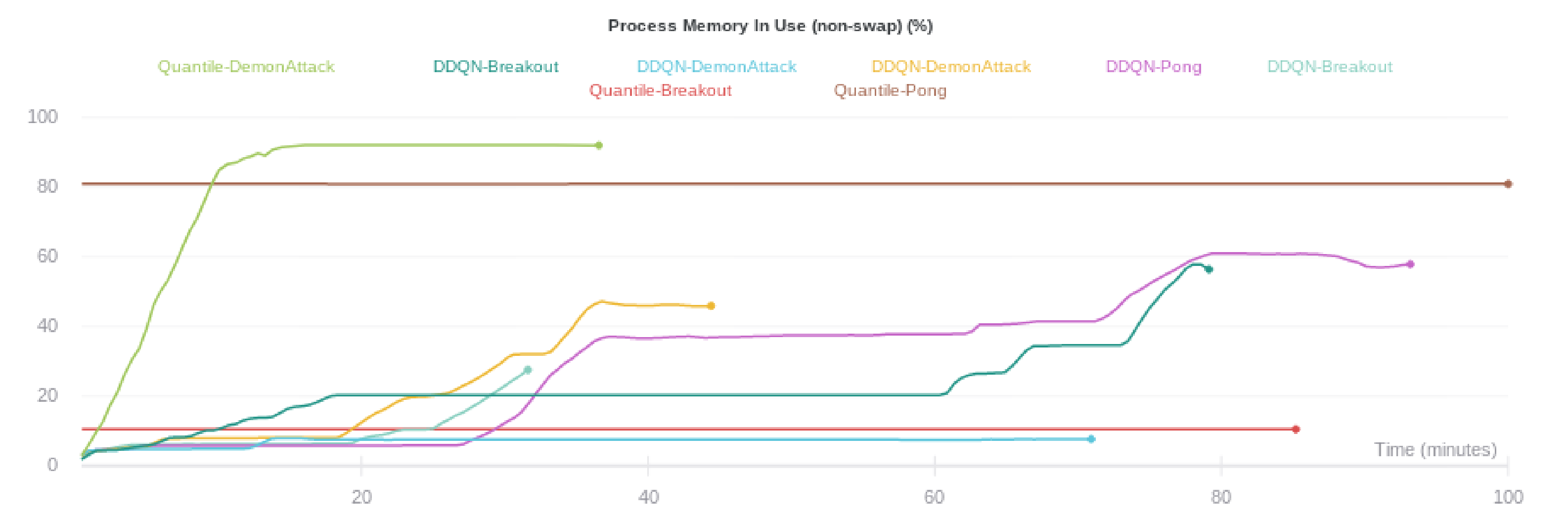
The mentioned challenges contribute to the biggest challenge, which was the fact that the model is very sensitive to hyperparameters. A small change in them can make the difference between the agent out-performing humans and failing completely. This is because hyperparameters change the way that the agent explores, which can lead it to not learn or burn in sub-optimal strategies. As mentioned in an article [Irpan, 2018], this makes it difficult to tell if the model is not learning because of software error, a hyperparameter error, or we just got unlucky with the seed.

Another challenge was the computational power at our disposal. Some of the games such as Yars Revenge only start learning after 50000 frames, as mentioned in the Rainbow paper [Hessel et al., 2017b]. This meant that we could not feasibly implement and debug the model on such games, as it would be very time consuming, hence we had to restrict our games to the simpler ones.

Lastly the $\mathbf{\Phi_z} d_t$ projection step in C51 was challenging to implement, noting that the same authors reference this fact in their follow up work [Dabney et al., 2017]; this can be largely attributed to the fact that the authors develop results for the $\bar{d}_p$ metric, which requires that two random variables share the same support. [Bellemare et al., 2017] vary the positions of diracs used to approximate the probability density over a fixed support of returns, [Dabney et al., 2017] tranpose this approach and instead use diracs to approximate a fixed cumulative density function over a varying support. The advantage of this is that no nearest neighbour interpolation is required to over come disjoint supports, reducing computational cost.
[Dabney et al., 2017] utilise a differentiable variant of the *huber* loss. This loss is an asymmetric convex loss function that penalizes the predicted quantiles $u = \hat{Z} - \theta$ for underestimation with weight $\frac{1}{2}u^2$ and overestimation with $k(|u| - \frac{1}{2}k)$ within the interval of estimation $[-k,k]$.

Quantile Regression Loss

$$\rho_\tau = \begin{cases} \frac{1}{2}u^2 & |u| < k \\ k(|u| - \frac{1}{2}k) & \text{otherwise} \end{cases} \tag{17}$$



(a) Graph of memory utilization of each agent in each game

Investigating Sample Inefficiency in Reinforcement Learning

Adrian Coutsoftides, Doddi El-Gabry, Vytenis Šliogeris and Alex Vavvas

University of Surrey

Implementation

We implement three models:

1. Dueling Double DQN + Prioritised Replay
2. Rainbow DQN with Quantile Regression

and measured their sample efficiency on a test-bed of three Atari games of similar difficulty, Pong, Breakout and Demon Attack. As a proxy for sample efficiency, we record the number of episodes it took each agent to "solve" the game. The requirements to solve each game depend on the game rules, testing across multiple environments allows us to gauge the overall effectiveness of the algorithm on heterogeneous tasks.

Within the limits of computational resources, we attempted to train each agent on each game for 100,000 frames or until the game is solved. The hyper-parameters are kept constant for all agents in order to accurately compare the performance of each algorithm:

Parameter	Value
Batch Size	64
Hidden Layer Size	512
ADAM Learning rate	$1e^{-5}$
Replay Buffer Size	30000
Priority β	$0.4 \to 1$
Priority α	0.4
Num. Quantiles	51
Target update frequency	1000
γ	0.99

Table: Table of hyperparameters

We also introduce a variation to the original Rainbow-DQN algorithm, as mentioned previously, we integrate the quantile regression loss in place of the distributional loss in the C51 algorithm. This is a straightforward extension, where the network is tasked with predicting the quantile functions of each action in place of its expected value. The value and advantage heads of the dueling network are repurposed to predict their respective quantile distributions instead of expected returns, this is shown in eq. (6). The Q values are calculate when selecting $\max_a Q(s,a)$, however the loss is directly applied to the predicted quantiles using the huber loss shown previously in order to correctly penalize the network for it's ability to predict quantiles as shown in eq. (7). As in the canonical implementation, we predict 51 quantiles, although there is no restriction on the resolution of the predicted distribution.

Integrating quantile returns

$$\begin{gathered} V(s) \coloneqq \theta_V \backsim Z_{\theta_V} \\ A(s,a) \coloneqq \theta_A^a \backsim Z_{\theta_A} \text{ where } a \in \{1, 2, \ldots, |\mathbf{A}|\} \\ Q(s,a) = \theta_V + (\theta_A^a - \mathbb{E}_{a \backsim \mathbf{A}}[\theta_A^a]) \end{gathered} \tag{18}$$

$$\nabla E_\theta = w_j \cdot \delta_j \cdot \mathbb{E}_{\hat{Z} \backsim Z}[\rho_\tau(\hat{Z} - \theta_i)] \tag{19}$$

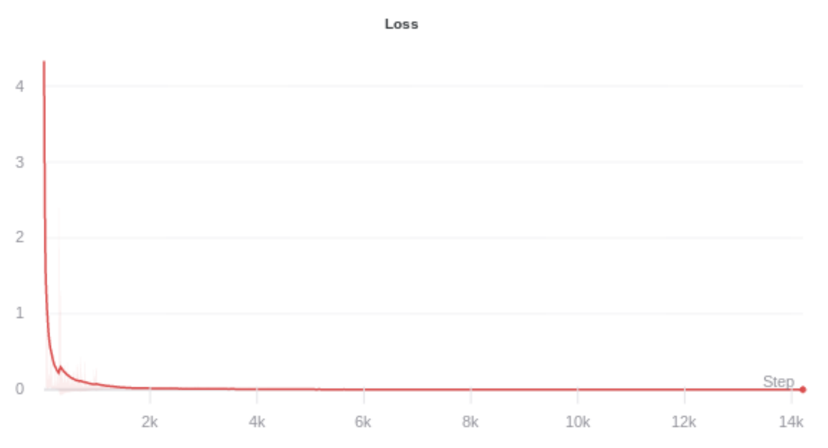
Equation 18 is the integration of dueling networks with quantile networks. Equation 19 is then integrating eq. 18 with PER. Implementing quantile distributions proved to be much more straightforward and expressive of the underlying distribution in practice when analysing experimental results.

Results & Evaluation

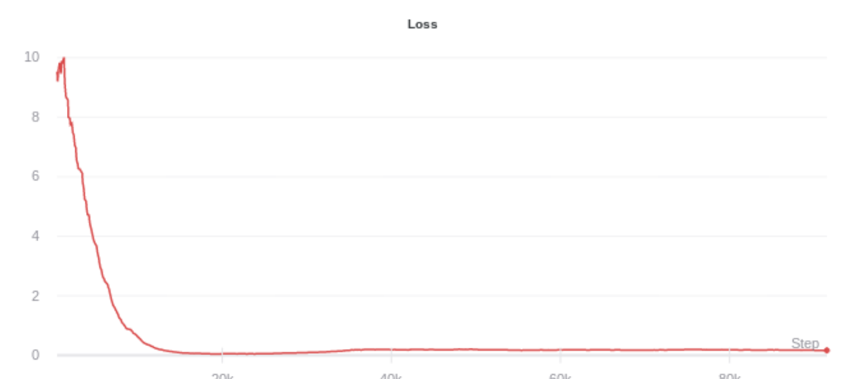
Over the course of testing, it was difficult to consistently train the models as they were prone to crashes due to memory error. This was because of the size of the experience replay buffer. In order to get the models to learn on more difficult games such as Breakout and Demon Attack, a larger pool of samples is required to learn the more complex policies of these games. In particular, the internal structure of the prioritised replay buffer is that of a *sum segment tree*, which is a binary heap data structure that maintains an array of data (in this case sampled transitions) that grows with space complexity $O(n)$ and another array of pointers to the nodes on the tree that grows at a rate of $O(2n)$ to maintain the binary relationships. While this enables faster time complexity when updating priorities on the tree $O(n \log n)$, it does incur a significant cost to memory, often crashing our GTX Titan Vs or overload RAM, as is evident by the figure on the previous page.

Considering that 4 stacked frames are preprocessed and passed into the model as a three dimensional matrix of shape $(4, 84, 84)$, the full memory size after 30000 transitions is approximately $\frac{(4*84*84*30000*64)}{8\times10^9} \approx 6.7\texttt{GB}$ assuming 64 bit floating point memory. In addition to the background load on the card, this proved enough to overwhelm the on-device memory, especially in the case of DemonAttack.

However in the case of pong, although DDQN was unable to converge, our Quantile-Rainbow agent converged significantly faster, within 70 episodes, indicating that the addition of n-step learning and distributional returns greatly helped the agent extract more information from its internal representation of the environment. Indeed the success appears to be replicated in Breakout as the agent's average rewards appear to begin increasing after 10000 timesteps, however this is an improper evaluation as we were unable to train DDQN for equally as long, due to aforementioned failures. Although we have strong results for the simple pong environment, more analysis is required.



(a) Graph of the loss for the DDQN model.



(b) Graph of the loss for the Quantile model.

Figure: Averaged loss along each game.

References

[Aytar et al., 2018] Aytar, Y., Pfaff, T., Budden, D., Paine, T. L., Wang, Z., and de Freitas, N. (2018). Playing hard exploration games by watching youtube.

[Bellemare et al., 2017] Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning.

[Dabney et al., 2017] Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2017). Distributional reinforcement learning with quantile regression.

[Fortunato et al., 2017] Fortunato, M., Gheshlaghi Azar, M., 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. *arXiv e-prints*, page arXiv:1706.10295.

[Hasselt, 2010] Hasselt, H. V. (2010). Double q-learning. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2613--2621. Curran Associates, Inc.

[Hessel et al., 2017a] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017a). Rainbow: Combining improvements in deep reinforcement learning.

[Hessel et al., 2017b] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. (2017b). Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298.

[Irpan, 2018] Irpan, A. (2018). Deep reinforcement learning doesn't work yet. `https://www.alexirpan.com/2018/02/14/rl-hard.html`.

[Mnih et al., 2015] 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.

[Sutton, 2018] Sutton, R. (2018). *Reinforcement learning : an introduction*. The MIT Press, Cambridge, Massachusetts London, England.

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

[Tishby and Zaslavsky, 2015] Tishby, N. and Zaslavsky, N. (2015). Deep learning and the information bottleneck principle.

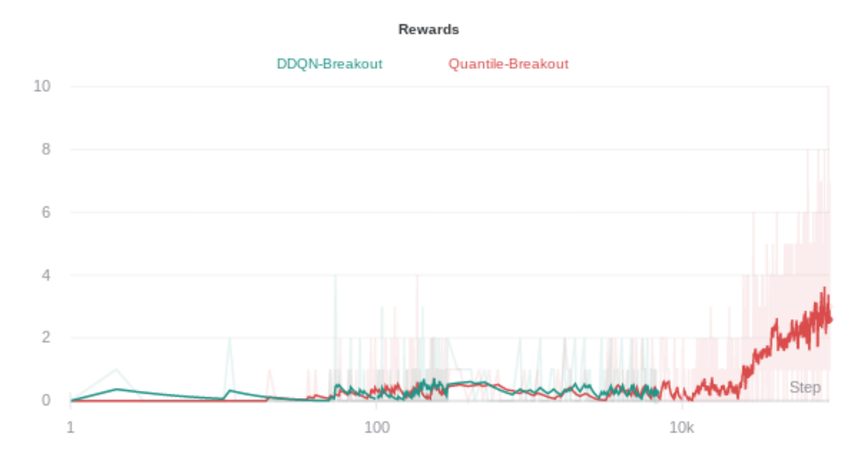
[Wang et al., 2015] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2015). Dueling network architectures for deep reinforcement learning.

Findings & Conclusions

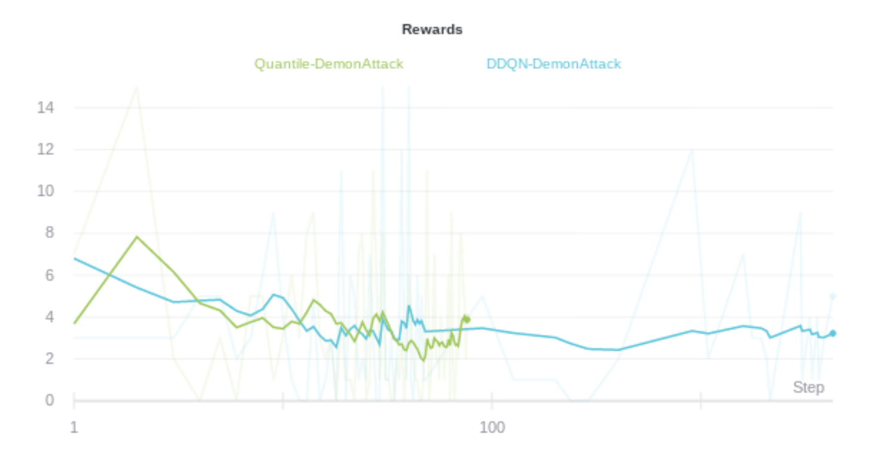
Despite our agent's strong results on Pong, the remaining results appear to be inconclusive even though Breakout looks promising. We ran multiple tests on each agent on each game, attempting to train them for 100,000 frames each. Often in the more complex environments the system resources would prove too much for the hardware at hand, echoing the verbalisms of [Irpan, 2018]. Although when we observed the recorded videos of each agent, we found that quantile agent did appear to be acting intelligently much earlier on than the DDQN agent, indicating that the quantile returns provided a strong training signal earlier on during the training process, improving sample efficiency drastically on Pong.

An interesting observation is that of the average loss graph over all models, there appears to be a consistent spike earlier on in the training data followed by convergence to a stable state. This appears to be in line with the work of [Tishby and Zaslavsky, 2015] demonstrating the *Information bottleneck principal*, this loss appears to initially spike as the neural network works to compress the representation of the input into an efficient embedding, followed by a second phase where the "network maximizes the conditional entropy of the layers subject to the empirical error constraint" [Tishby and Zaslavsky, 2015]. This is inline with our hypothesis indicating that one of the primary bottlenecks to sample efficiency is in fact the efficacy of the neural network's ability to extract useful information from the representation, as the representation itself is already rich but the information must be extracted with sufficient sophistication, but not so much as to introduce excess bias.

In conclusion, although the claims of [Irpan, 2018] proved to hold true in practise, there is great promise in new techniques that extract more useful information from the the neural network's internal representations. For follow-up work we would further investigate this link between the *Information bottleneck principal* and evaluate our results on a wider array of environments.

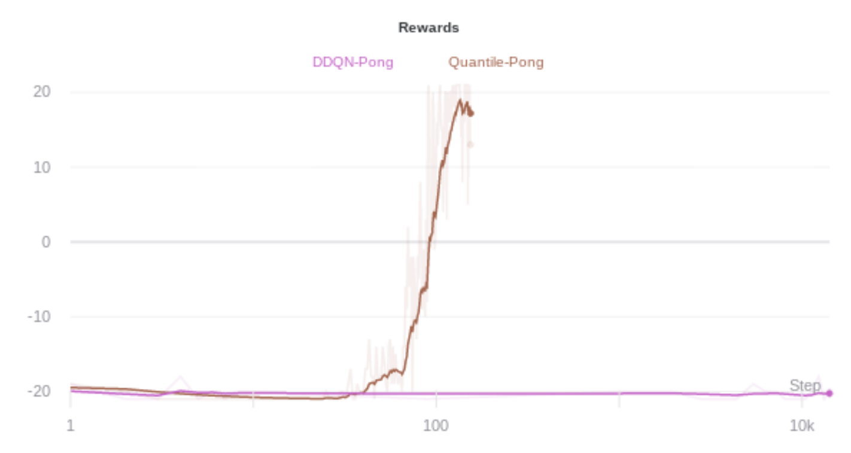


(a) The rewards of the DDQN and Quantile agents training on Breakout.

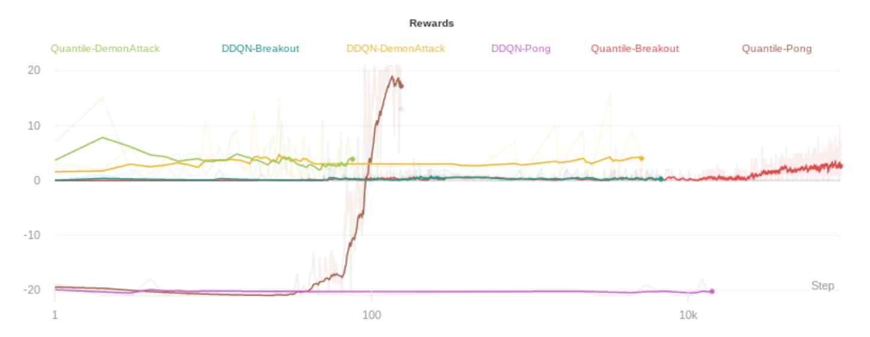


(b) The rewards of the DDQN and Quantile agents training on Demon Attack.

Figure: The rewards of the DDQN and Quantile agents training on Breakout and Demon Attack.

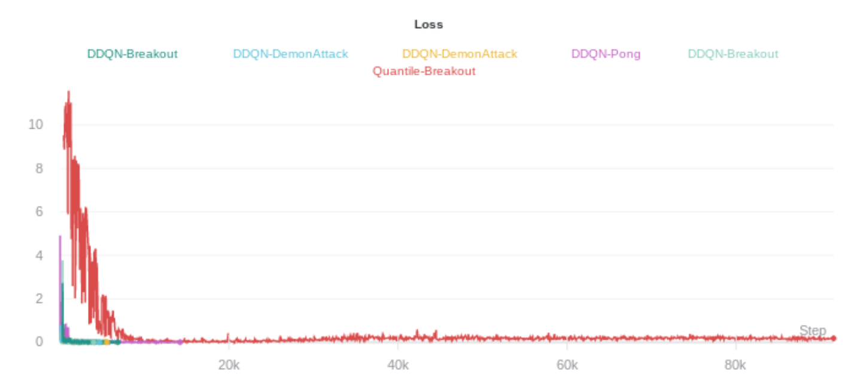


(a) The rewards of the DDQN and Quantile agents training on Pong.

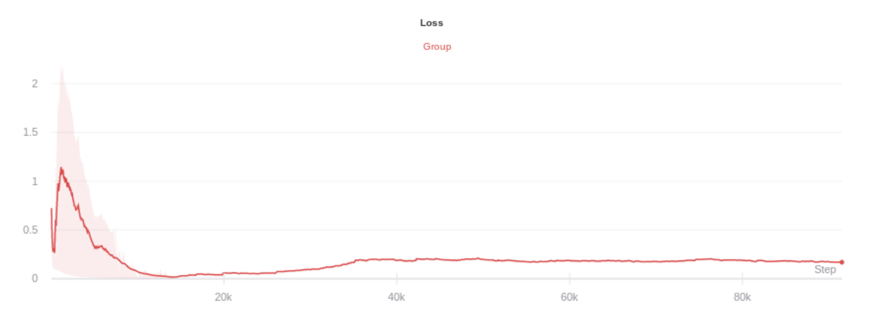


(b) The reward of each model training on each game.

Figure: The reward of each of the models training on Pong, and a graph of all of the reward functions.



(a) Graph of the loss against frames trained for each of the models on each of the games.



(b) Average loss of all models and all games. Shaded area signifies one standard deviation.

Figure: Graphs of the loss against frames trained.