Eligibility Traces

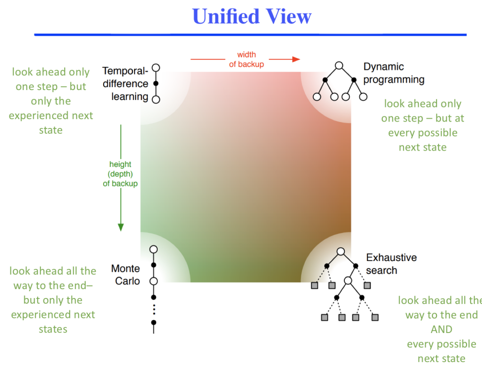


Figure 1: A unified view - The Long-term of AI & Temporal-Difference Learning – Rich Sutton

Family of methods between Temporal Difference & Monte Carlo

Eligibility traces allow us to **assign TD errors** to different states - can be useful with delayed rewards or non-Markov environments - requires more computation - squeezes more out of data

Allow us to tradeoff between bias and variance

The space between TD and MC

In between TD and MC exist a family of approximation methods known as **n-step returns**

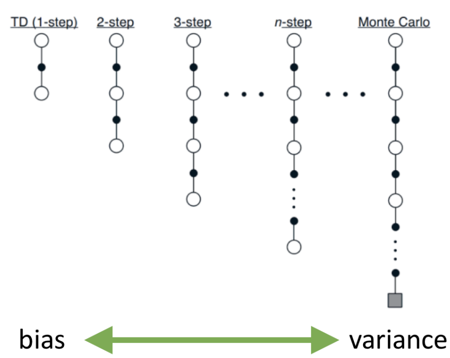


Figure 2: Sutton & Barto

Forward and backward view

We can look at eligibility traces from two perspectives

The **forward** view is helpful for understanding the theory

The **backward** view can be put into practice

We can decompose return into **complex backups** - looking forward to future returns - can use a combination of experience based and model based backups

$$R\_t = \frac{1}{2}R\_t^2 + \frac{1}{2}R\_t^4$$

$$R\_t = \frac{1}{2}TD + \frac{1}{2}MC$$

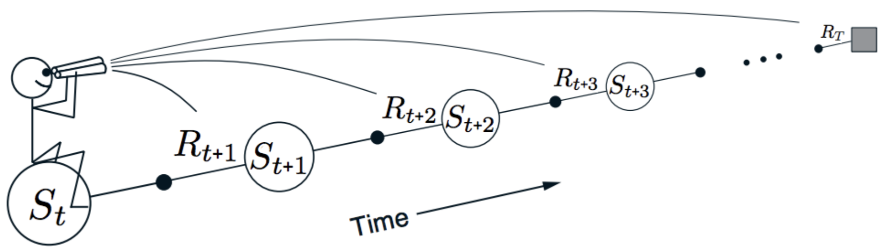


Figure 12.4: The forward view. We decide how to update each state by looking forward to future rewards and states.

Figure 3: Sutton & Barto

The backward view

The backward view approximates the forward view - forward view is not practical (requires knowledge of the future)

It requires an additional variable in our agents memory - **eligibility trace** $e_t(s)$

At each step we decay the trace according to

$$e\_t(s) = \gamma\lambda e\_t - 1(s)$$

Unless we visited that state, in which case we accumulate more eligibility

$$e\_t(s) = \gamma\lambda e\_t - 1(s) + 1$$

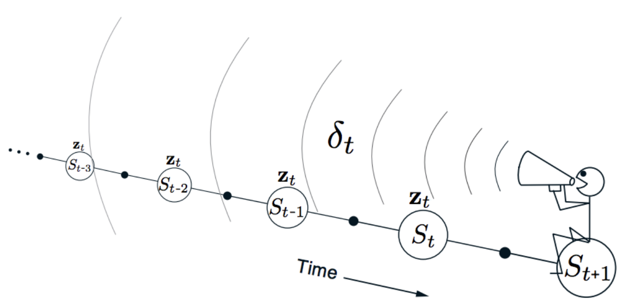
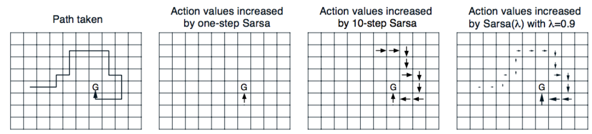


Figure 12.5: The backward or mechanistic view. Each update depends on the current TD error combined with the current eligibility traces of past events.

Figure 4: Sutton & Barto

Example 12.1: Traces in Gridworld The use of eligibility traces can substantially increase the efficiency of control algorithms over one-step methods and even over n-step methods. The reason for this is illustrated by the gridworld example below.



The first panel shows the path taken by an agent in a single episode. The initial estimated values were zero, and all rewards were zero except for a positive reward at the goal location marked by G. The arrows in the other panels show, for various algorithms, which action-values would be increased, and by how much, upon reaching the goal. A one-step method would increment only the last action value, whereas an n-step method would equally increment the last n action's values, and an eligibility trace method would update all the action values up to the beginning of the episode to different degrees, fading with recency. The fading strategy is often the best tradeoff, strongly learning how to reach the goal from the right, yet not as strongly learning the roundabout path to the goal from the left that was taken in this episode. ■

Figure 5: Sutton & Barto

Traces in a grid world

- one step method would only update the last $Q(s,a)$
- n-step method would update all $Q(s,a)$ equally
- eligibility traces updates based on how recently each $Q(s,a)$ was experienced

Prioritized experience replay

Published as a conference paper at ICLR 2016

PRIORITIZED EXPERIENCE REPLAY

Tom Schaul, John Quan, Ioannis Antonoglou and David Silver
Google DeepMind
{schaul,johnquan,ioannisa,davidsilver}@google.com

Figure 6: fig

Naive experience replay

Naive experience replay randomly samples experience - learning occurs at the same frequency as experience

Prioritized Experience Replay

Some experience is more useful for learning than others - we can measure how useful experience is by the temporal difference error

$$error = r + \gamma Q(s',a) - Q(s,a)$$

TD error measures suprise - this transition gave a higher or lower reward than our value function expected

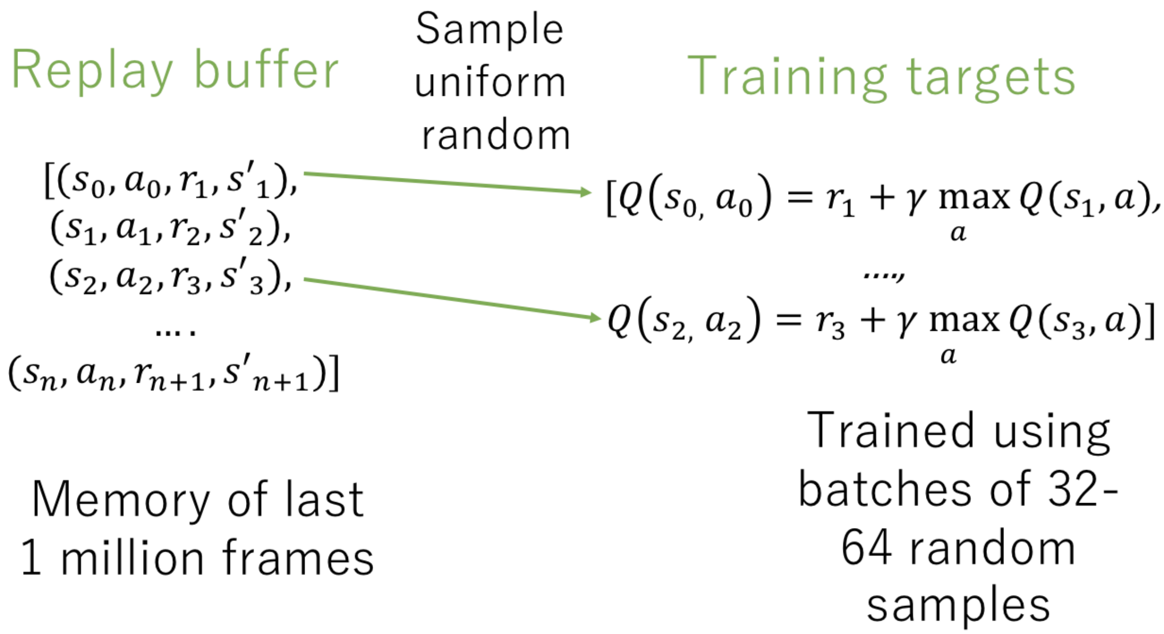


Figure 7: fig

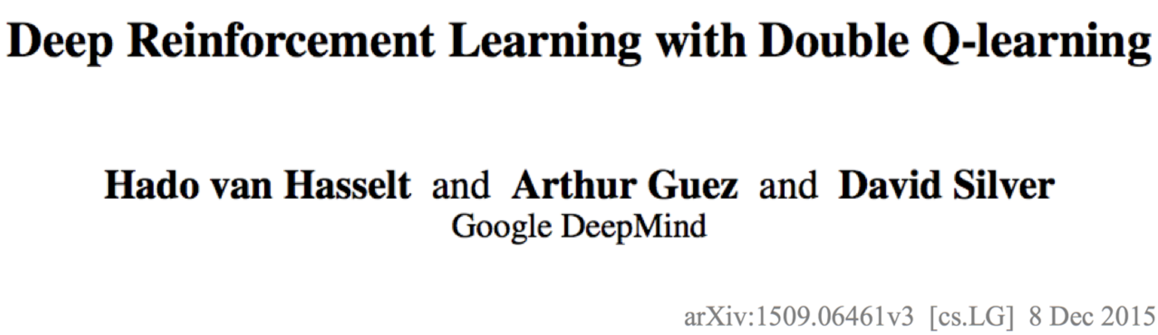
Non-random sampling introduces two problems

1. loss of diversity - we will only sample from high TD error experiences
2. introduce bias - non-independent sampling

Schaul et. al (2016) solves these problems by

1. loss of diversity -> make the prioritization stochastic
2. correct bias -> use importance sampling

DDQN


Deep Reinforcement Learning with Double Q-learning

Hado van Hasselt and **Arthur Guez** and **David Silver**
Google DeepMind

arXiv:1509.06461v3 [cs.LG] 8 Dec 2015

Figure 8: fig

DDQN = Double Deep Q-Network - first introducued in a tabular setting in 2010 - reintroduced in the content of DQN in 2016

DDQN aims to overcome the **maximization bias** of Q-Learning

Maximization bias

Imagine a state where $Q(s, a) = 0$ for all a

Our estimates are normally distributed above and below 0

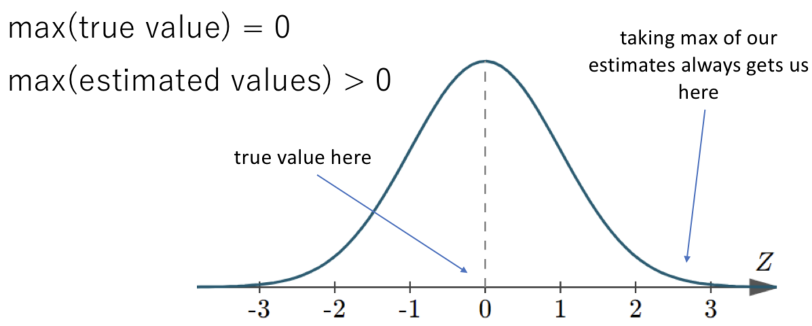


Figure 9: fig

The DDQN modification to DQN makes use of the target network as a different function to approximate Q(s,a)

Original DQN target

$$r + \gamma \max_a Q(s, a; \theta^-)$$

DDQN target

$$r + \gamma Q(s', \underset{a}{argmax} Q(s', a; \theta); \theta^-)$$

- select the action according to the online network
- quanitfy the value that action using the target network

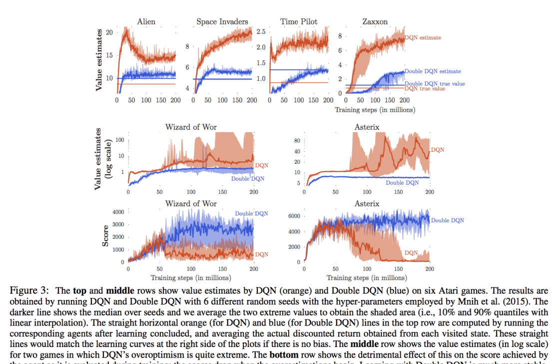
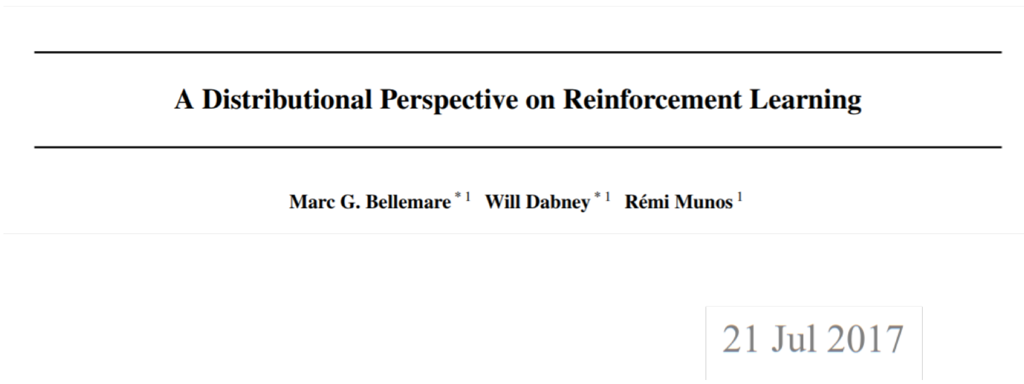


Figure 10: fig



A Distributional Perspective on Reinforcement Learning

Marc G. Bellemare [*1] Will Dabney [*1] Rémi Munos [1]

21 Jul 2017

Figure 11: fig

Distributional Q-Learning

Beyond the expectation

All the reinforcement learning we have seen focuses on the expectation (i.e. the mean)

$$Q(s,a) = \mathbf{E}[G_t] = \mathbf{E}[r + \gamma Q(s',a)]$$

In 2017 DeepMind introduced the idea of the value distribution

State of the art results on Atari (at the time - Rainbow is currently SOTA)

Figure 12: fig

Rainbow

All the various improvements to DQN address different issues

- DDQN - overestimation bias
- prioritized experience replay - sample efficiency
- dueling - generalize across actions
- multi-step bootstrap targets - bias variance tradeoff
- distributional Q-learning - learn categorical distribution of $Q(s,a)$
- noisy DQN - stochastic layers for exploration

Rainbow combines these improvements

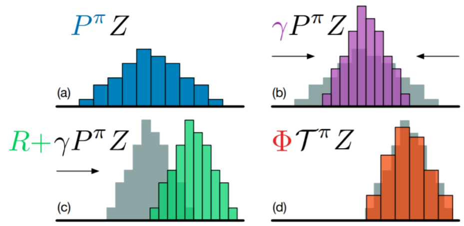


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section 4).

Figure 13: Bellamare et. al 2017

	Mean	**Median**	**> H.B.**	**> DQN**
DQN	228%	79%	24	0
DDQN	307%	118%	33	43
DUEL.	373%	151%	37	50
PRIOR.	434%	124%	39	48
PR. DUEL.	592%	172%	39	44
C51	**701%**	**178%**	**40**	**50**
UNREAL†	880%	250%	-	-

Figure 6. Mean and median scores across 57 Atari games, measured as percentages of human baseline (H.B., Nair et al., 2015).

† The UNREAL results are not altogether comparable, as they were generated in the asynchronous setting with per-game hyperparameter tuning (Jaderberg et al., 2017).

Figure 14: Bellamare et. al 2017

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

(Submitted on 6 Oct 2017)

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.

Figure 15: fig

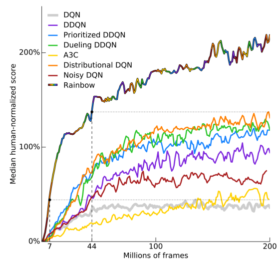


Figure 1: **Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final performance. Curves are smoothed with a moving average over 5 points.

Figure 16: fig

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).

Figure 17: fig

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 18: fig

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 litera-

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	164%	125%
Rainbow	223%	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 19: fig