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*

Presenter: Sridhar Reddy Velagala

What is Deep RL?

RL (Reinforcement learning)

- Come up with the best action given a state
- Maximize reward

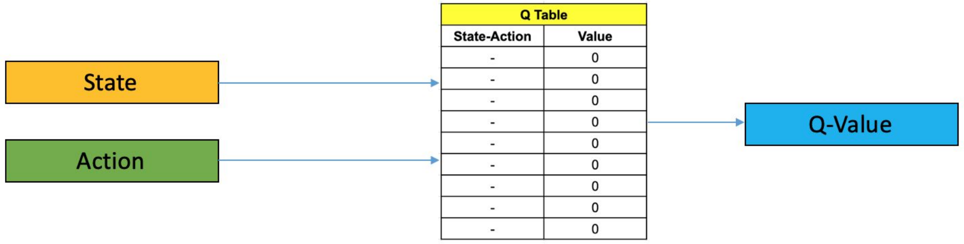
Deep RL

- Use neural networks to make that decision for action

Q learning is a “model free” RL algorithm to learn the value of an action at a particular state.

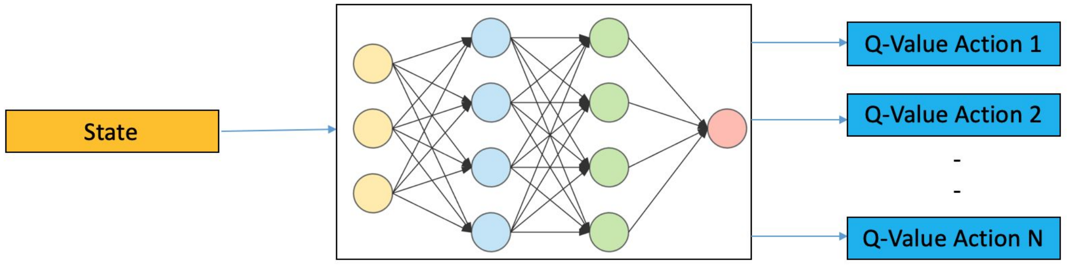
What is DQN (Deep Q Networks)?

Q learning is a “model free” RL algorithm to learn the value of an action at a particular state. Q(s, a)



State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q Learning

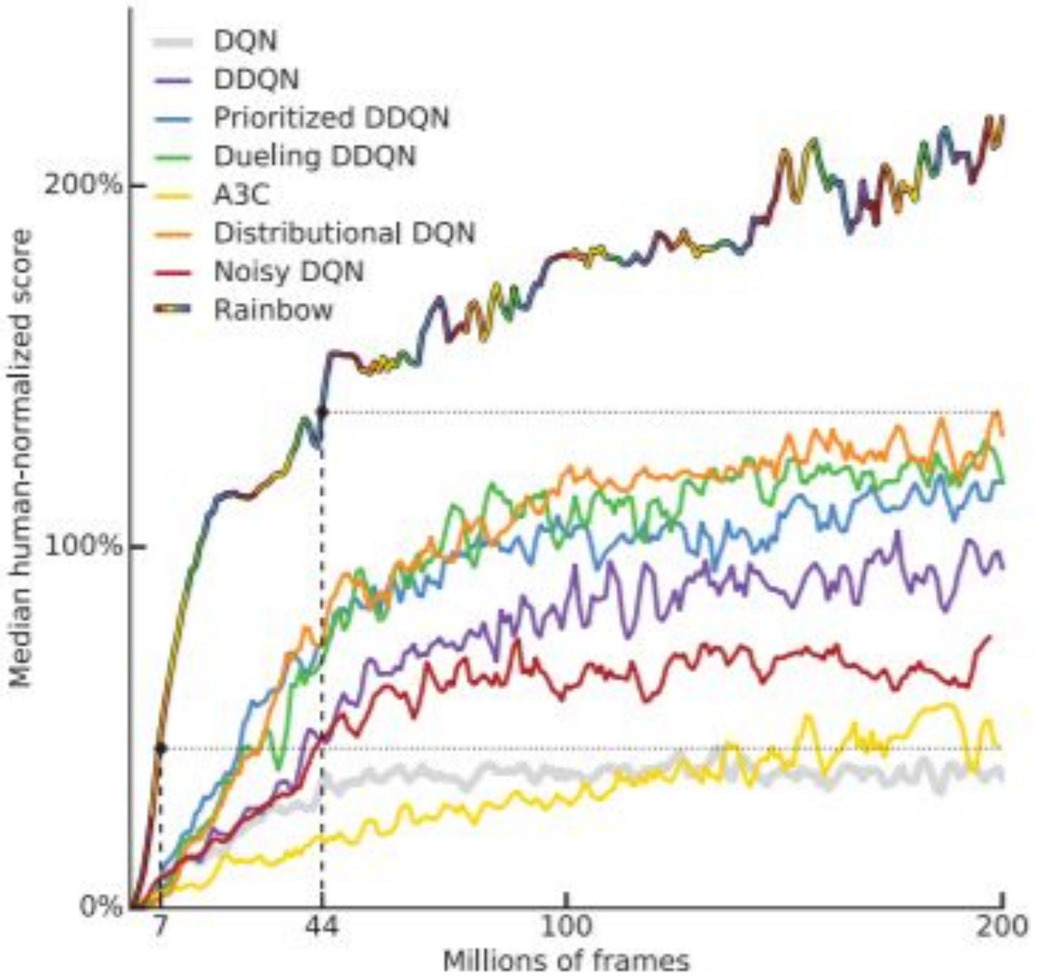


Deep Q Learning

What is this paper about?

DQN has extensions

Rainbow: Combining Improvements in Deep Reinforcement Learning



Extensions to DQN - Deep Q learning

DDQN - Double DQN

Prioritized DDQN

Dueling DDQN

A3C - Asynchronous Advantage Actor Critic (or) Asynchronous Actor Critic Agent

Distributional DQN

Noisy DQN

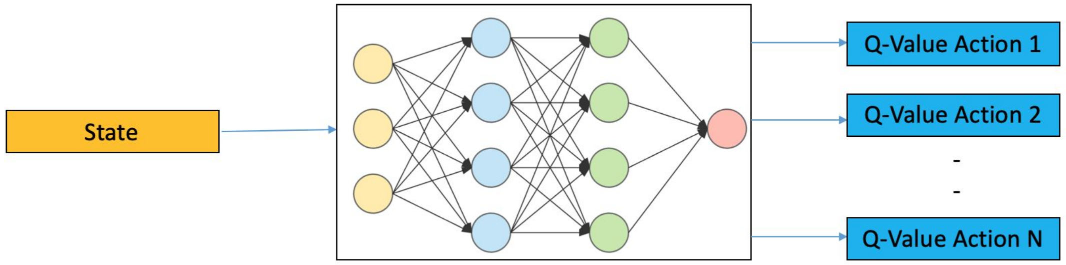
=> All improvements combined into the RAINBOW

How DQN works?

State is the image pixels of the environment

Outputs are the Q values for every possible action

We train this network in order to maximize the future possible reward

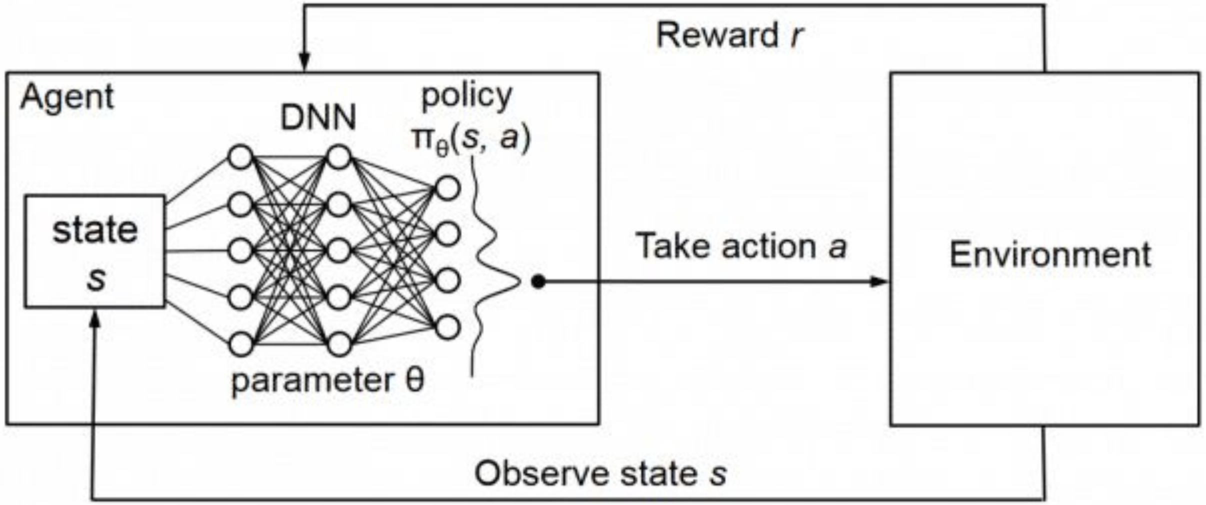


Deep Q Learning

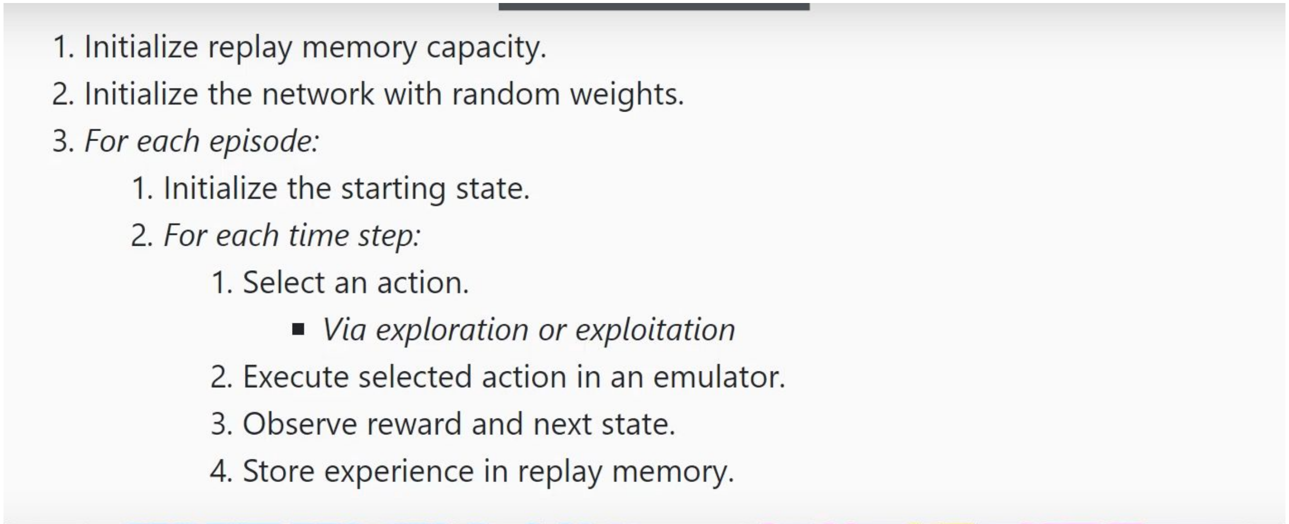
How DQN works?

We take a random action, based on the output action from the neural network

Reward R, Next state S_{t+1} are observed and stored in memory for training the network: **Replay Memory**



Training process



1. Initialize replay memory capacity.
2. Initialize the network with random weights.
3. *For each episode:*
    1. Initialize the starting state.
    2. *For each time step:*
        1. Select an action.
            - *Via exploration or exploitation*
        2. Execute selected action in an emulator.
        3. Observe reward and next state.
        4. Store experience in replay memory.

Exploration vs Exploitation

Explore the environment and select a random action

Exploit the environment and greedily select the action with the highest value

Based on probability p: 1 -> exploration, 0 -> exploitation (epsilon-greedy strategy)

Initially starts at p = 1 but is gradually lowered by some rate factor

Markov Decision Processes

$\langle S, A, T, r, \gamma \rangle$

S : Finite set of states.

A : Finite set of actions.

$T(s, a, s') = P[s_{t+1} = s' \mid S_t = s, A_t = a]$: (stochastic) Transition function

$r(s, a) = E[R_{t+1} \mid S_t = s, A_t = a]$: reward function.

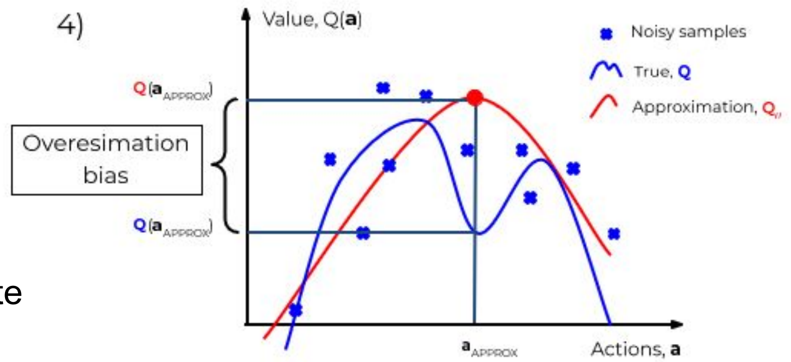
$\gamma \in [0,1]$: Discount factor

Double Deep Q- learning (DDQN)

Motivation

- Fix overestimation bias

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

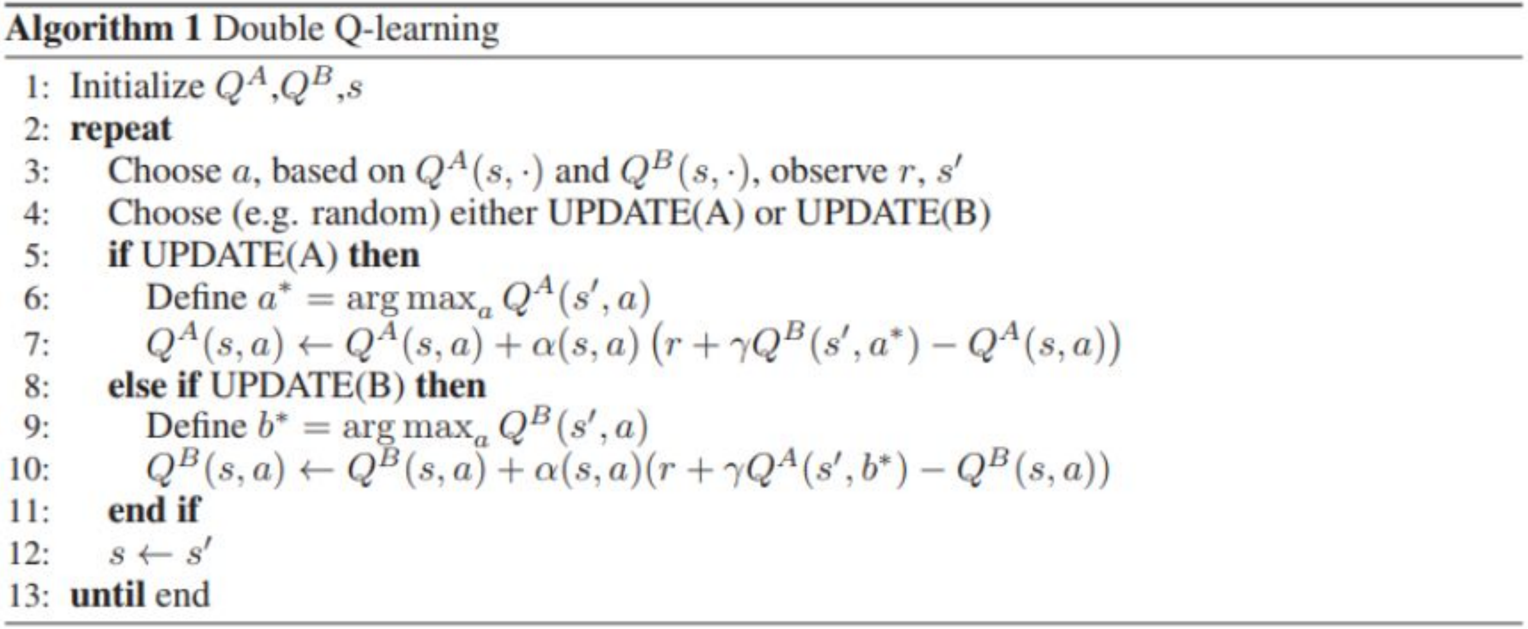


Procedure

- Reduce this bias by using two q-value functions that update each other and by using the following loss function

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

Double Deep Q- learning (DDQN)



Algorithm 1 Double Q-learning

```
1: Initialize $Q^A$,$Q^B$,$s$
2: **repeat**
3:     Choose $a$, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe $r$, $s'$
4:     Choose (e.g. random) either UPDATE(A) or UPDATE(B)
5:     **if** UPDATE(A) **then**
6:         Define $a^* = \arg\max_a Q^A(s',a)$
7:         $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
8:     **else if** UPDATE(B) **then**
9:         Define $b^* = \arg\max_a Q^B(s',a)$
10:        $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
11:    **end if**
12:    $s \leftarrow s'$
13: **until** end
```

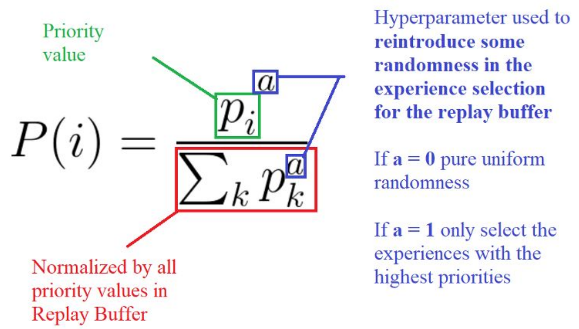
Algorithm taken from Double Q-learning by Hado van Hasselt

Quiz: How can we improve sample efficiency?

Prioritised DDQN

Motivation

- Improve sampling efficiency



$$P(i) = \frac{\boxed{p_i}^{\boxed{a}}}{\boxed{\sum_k p_k^{\boxed{a}}}}$$

Priority value

Hyperparameter used to **reintroduce some randomness in the experience selection for the replay buffer**

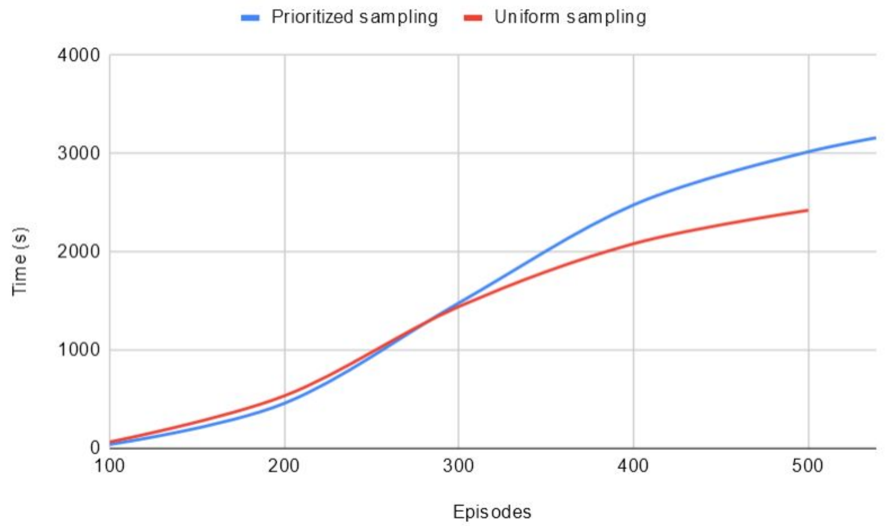
If **a = 0** pure uniform randomness

If **a = 1** only select the experiences with the highest priorities

Normalized by all priority values in Replay Buffer

Procedure

- New transitions get maximum priority
- Highest TD error



Dueling DDQN

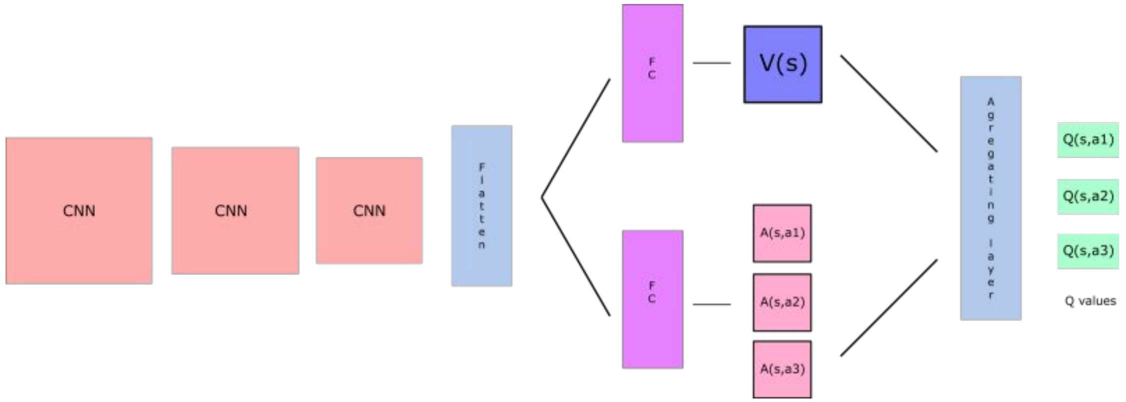
Motivation

- Give less importance to states where action is irrelevant

Procedure

- Features two streams: value and advantage

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$



Dueling DDQN

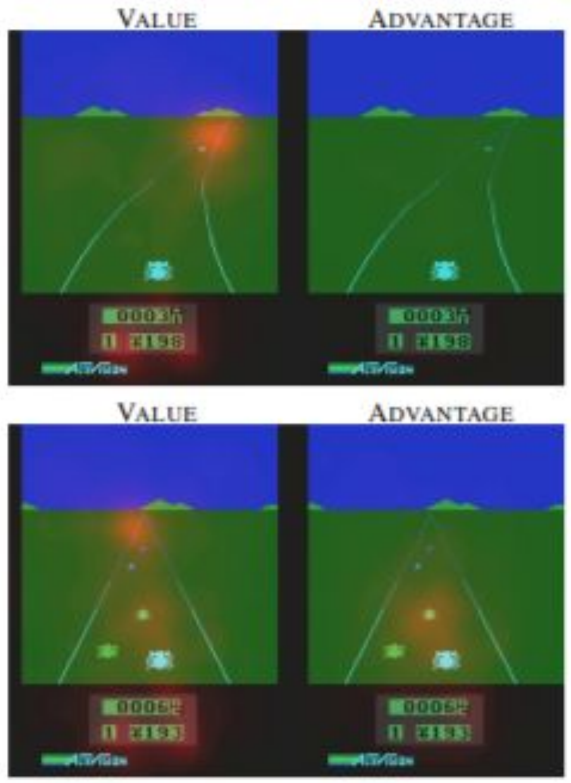


Figure 2. See, attend and drive: Value and advantage saliency maps (red-tinted overlay) on the Atari game Enduro, for a trained dueling architecture. The value stream learns to pay attention to the road. The advantage stream learns to pay attention only when there are cars immediately in front, so as to avoid collisions.

A3C - Multi step learning

Motivation

- Learn faster

Procedure

- Use forward view multi step targets.

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

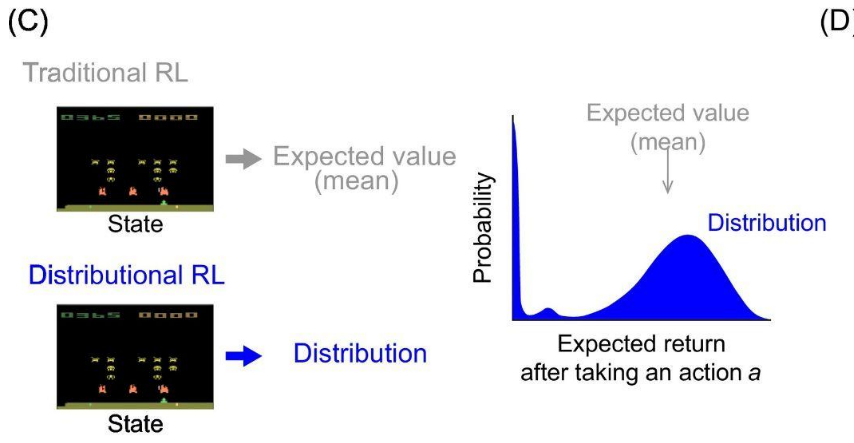
Distributional DDQN

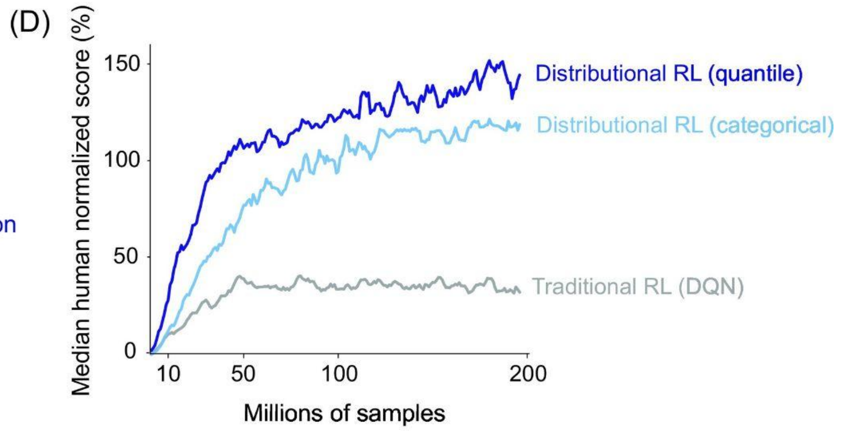
Motivation

- Using scalar values for action can be inaccurate and noisy

Procedure

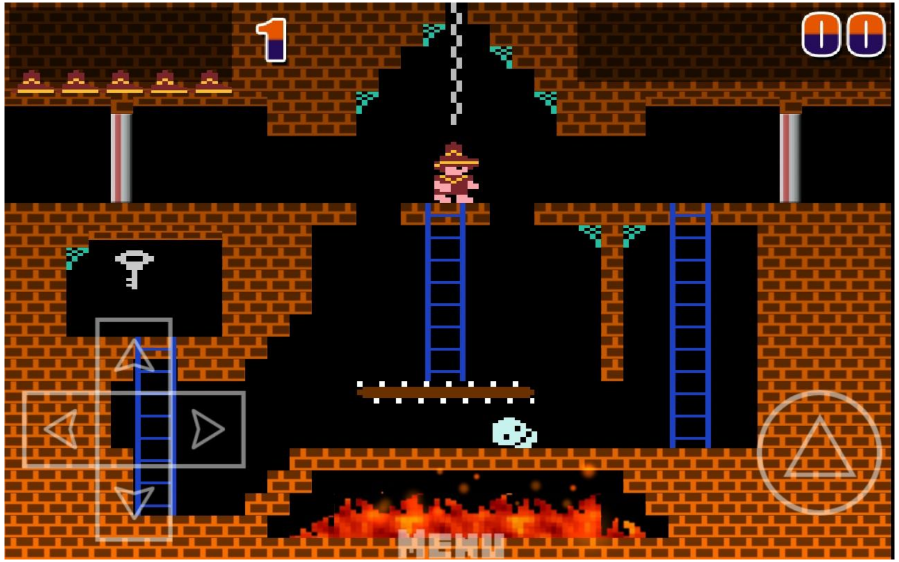
- Use a distribution instead (distributional bellman equation)





Trends in Neurosciences

Quiz: How can we improve on the epsilon greedy algorithm in situations where multiple actions needed to be performed to collect a reward?



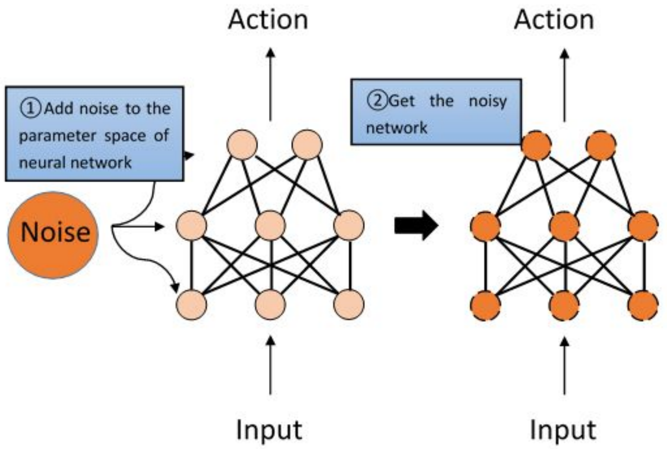
Noisy DDQN

Motivation

- Many actions needed to collect final reward
- Example: Montezuma's Revenge

Procedure

- Update the weights as **W = Mu + Sigma * epsilon**
    - Mu is a Variable with a random initialization
    - Sigma is a Variable with a constant initialization
    - epsilon is actually the noise (0,1)

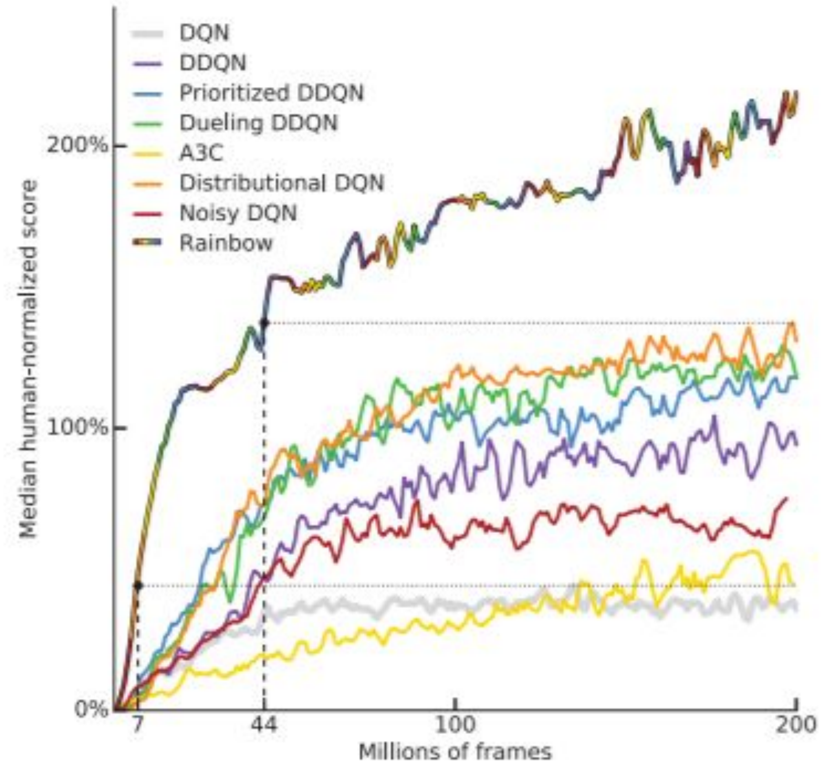


Evaluation

57 Atari 2600 games

- Average scores after every million steps

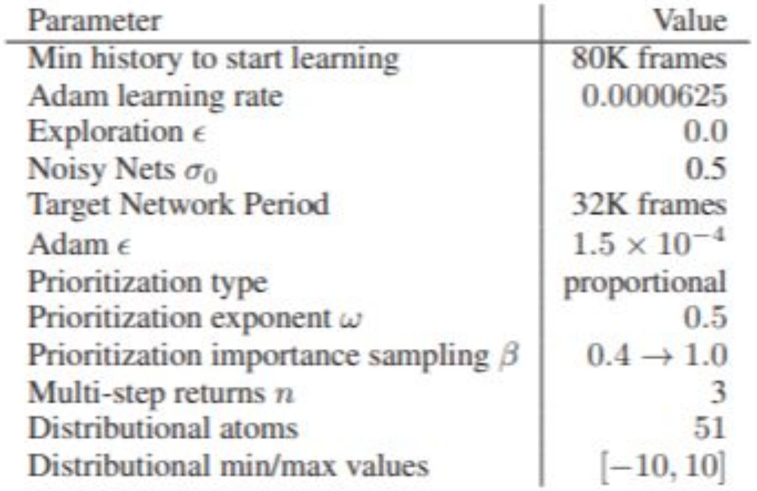
- Episodes limited to 108K frames (~ 30 mins of simulated play)



Discussion

Tuned most sensitive hyper parameters using manual coordinate descent

The hyper parameters are identical across all 57 games:

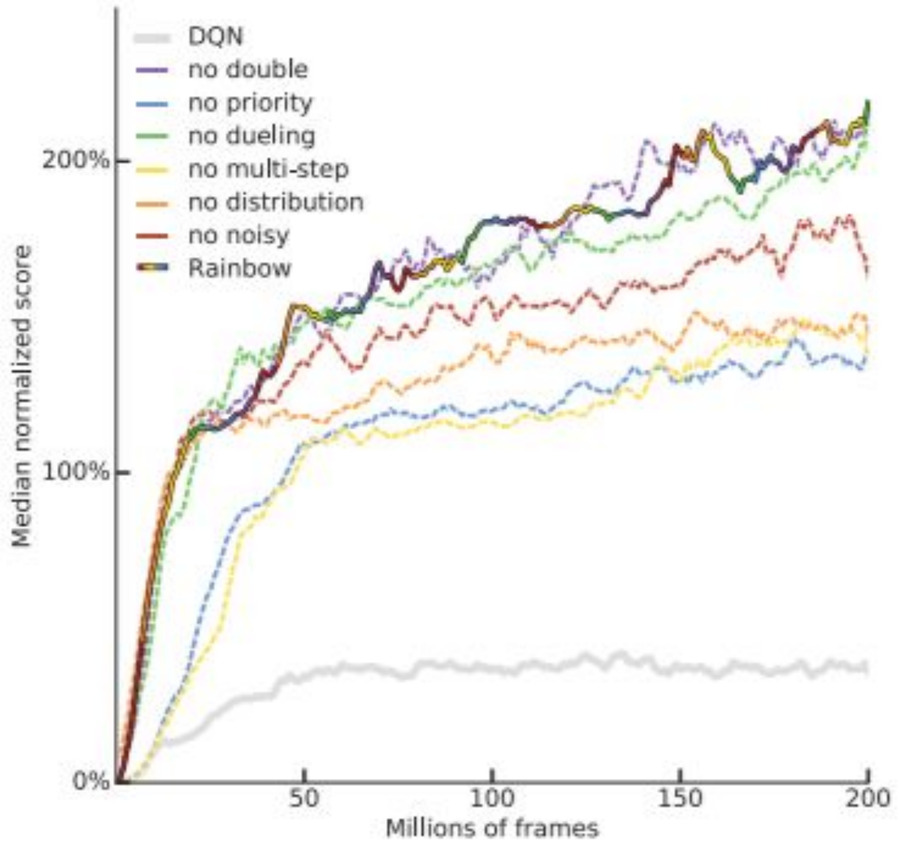


Parameter	Value
Min history to start learning	80K frames
Adam learning rate	0.0000625
Exploration ϵ	0.0
Noisy Nets σ_0	0.5
Target Network Period	32K frames
Adam ϵ	1.5×10^{-4}
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Multi-step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

Table 1: Rainbow hyper-parameters

Discussion

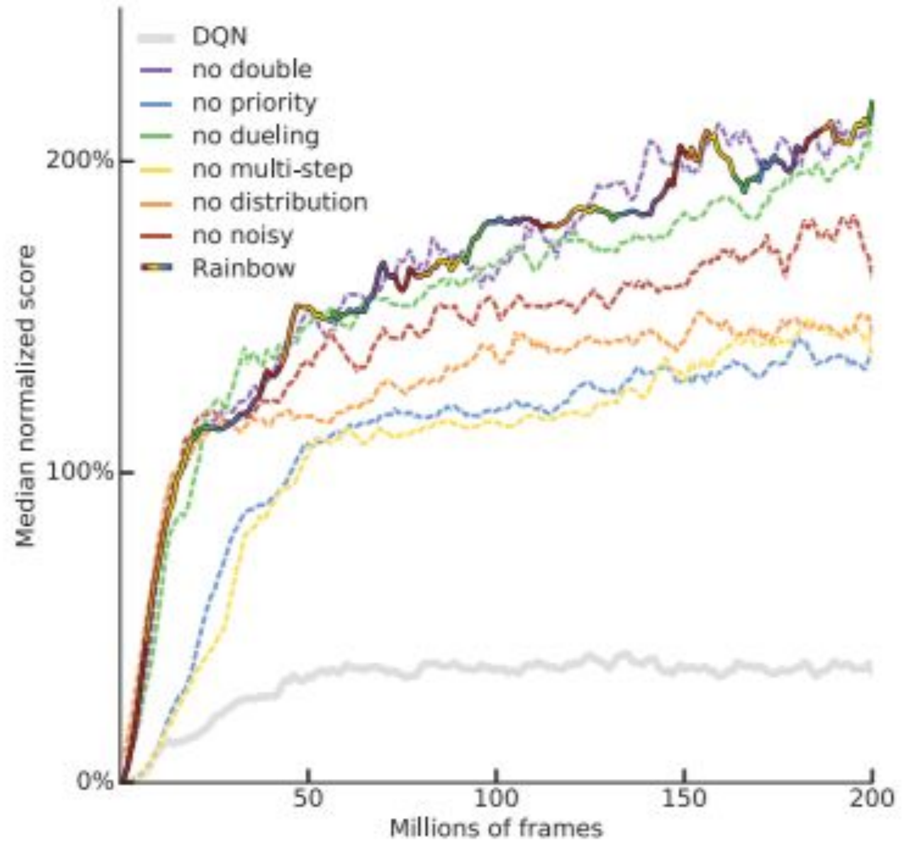
Quiz: Which algorithms have the most effect on performance?



Discussion

Multi step, prioritized DDQN has a large drop which also hurts early performance

Distributional DDQN has more impact in the far-later stages of the learning process



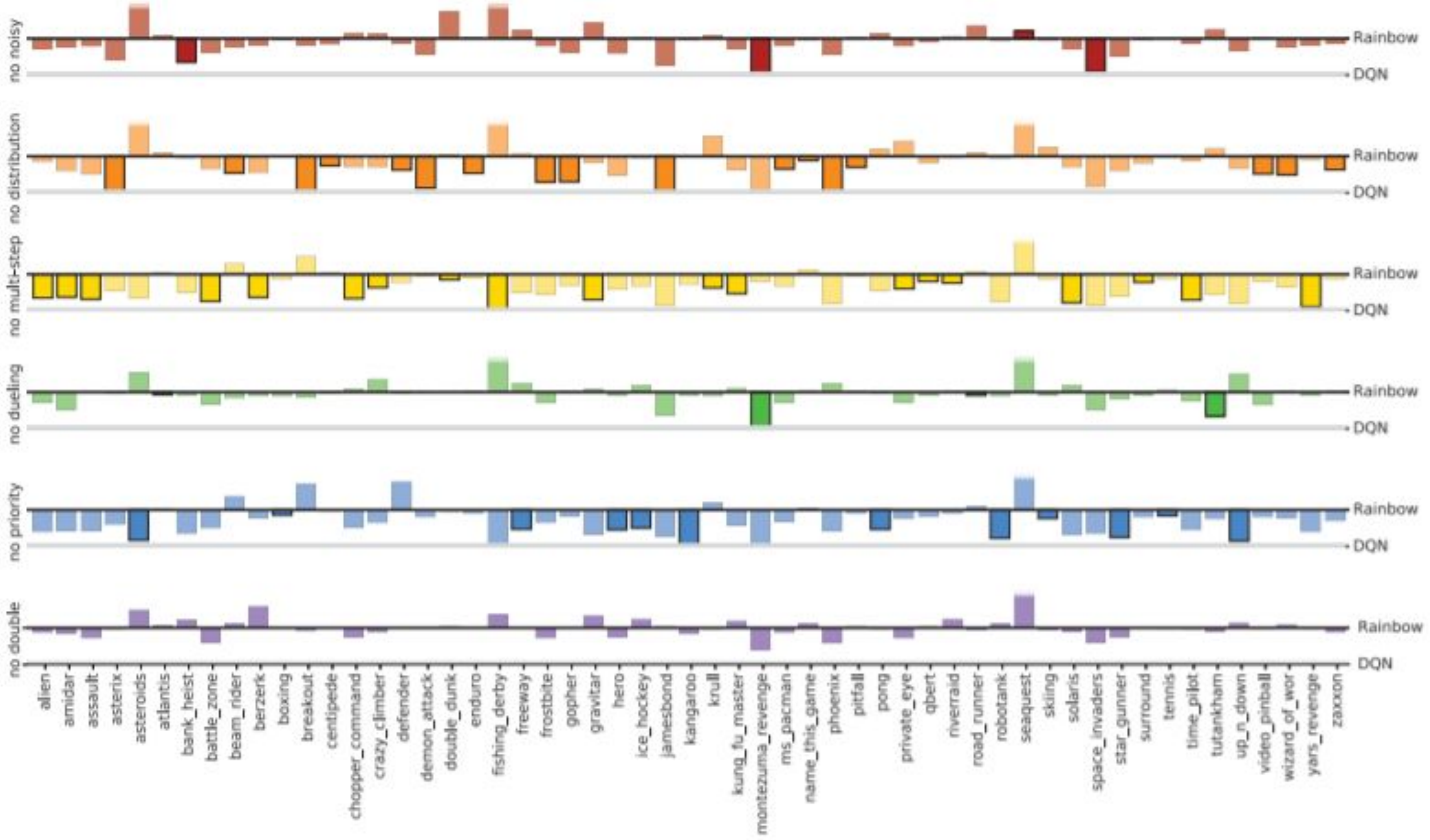


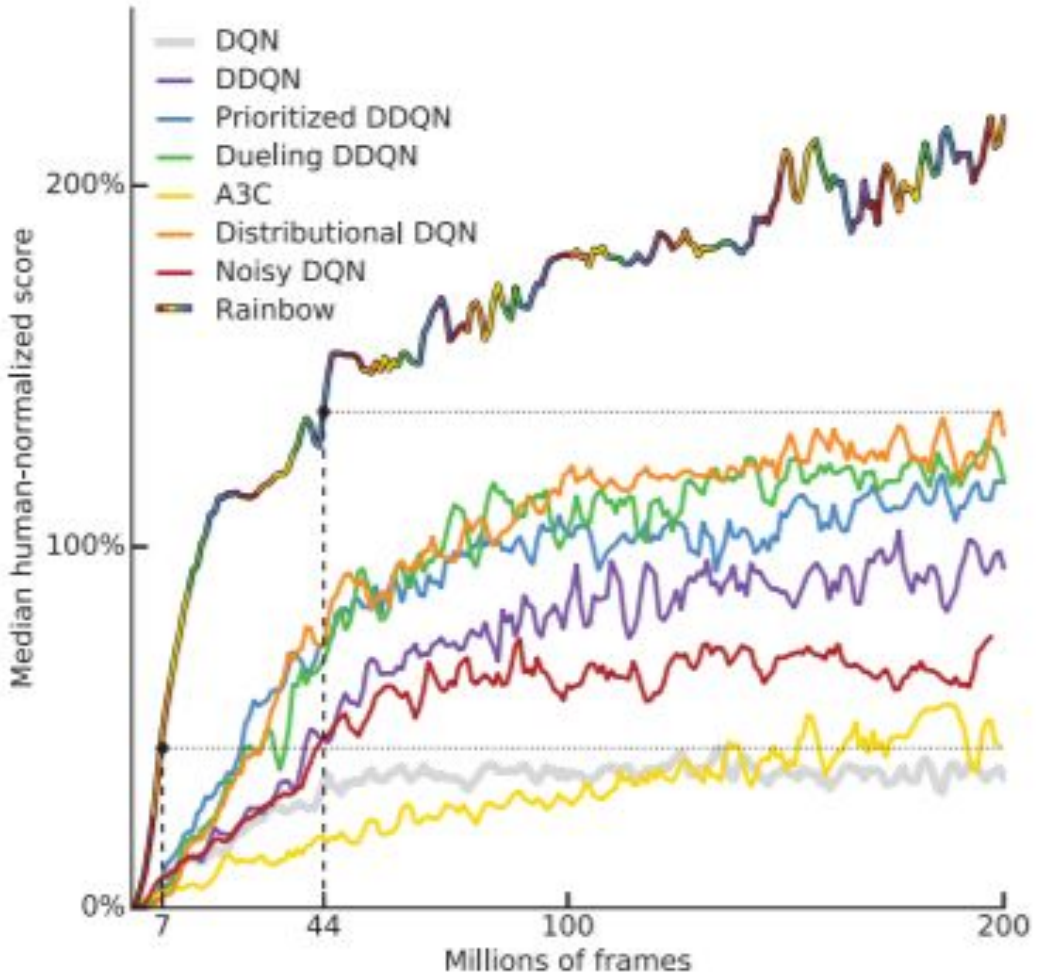
Figure 4: Performance drops of ablation agents: Performance is area under the curve, normalized relative to Rainbow and DQN. Venture and Bowling, where DQN outperformed Rainbow, are omitted. Ablations leading to the largest drop are highlighted per game. Prioritization and multi-step help almost uniformly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

Learning speed

7M frames = ~10 hrs real time

200M for the whole RUN = ~10 days

Parallelism -> future work



Future steps

Many potential methods were not considered

- TRPO, Actor critic methods
- Episodic control
- Bootstrapped DQN
- Count-based exploration
- Intrinsic motivation
- Asynchronous learning
- Evolution strategies
- Hierarchical RL and others…

Questions?

Comparison and ablation study

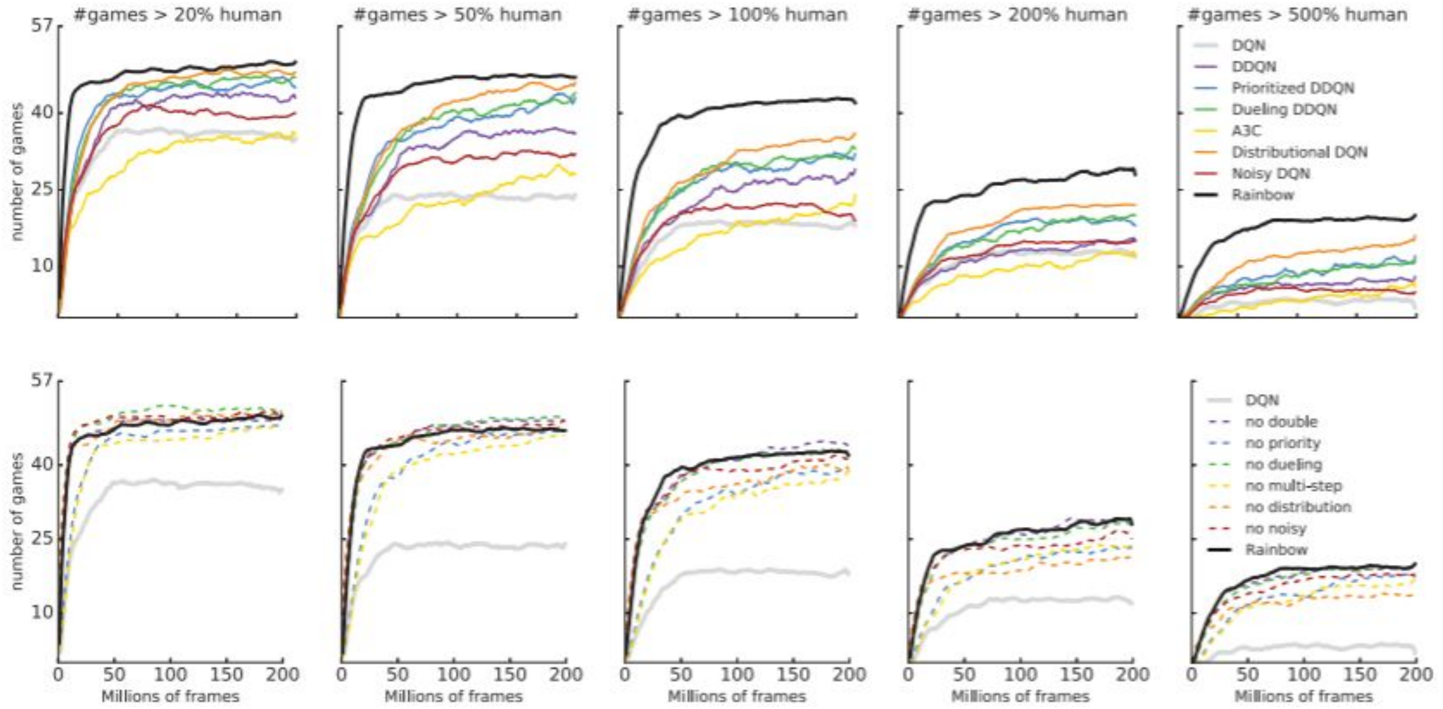


Figure 2: Each plot shows, for several agents, the number of games where they have achieved at least a given fraction of human performance, as a function of time. From left to right we consider the 20%, 50%, 100%, 200% and 500% thresholds. On the first row we compare Rainbow to the baselines. On the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

Integrated agent

0th distributional

1st multi step variant

2nd double q learning

3rd prioritized replay: TD(temporal difference) error -> KL loss - more robust to noisy stochastic environments

4th dueling

5th noisy

Markov decision Processes

Infinite horizon MDPs

- Tasks that go on forever e.g: running, juggling

*Episodic MDPs

- Has an end. Agent resets after every episode e.g: solving a maze, swinging a bat

Markov decision Processes

*Episodic MDPs

- Has an end. Agent resets after every episode e.g: solving a maze, swinging a bat

Assuming constant $\gamma_t = \gamma$, except for episode termination $\gamma_t = 0$

Agents side:

- Action selection is given by a policy π
- Discounted return is calculated $G_t = \sum_{k=0}^{\infty} \gamma_t^{(k)} R_{t+k+1}$
    - Discounted sum of future rewards collected by the agent

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 a given state action pair $q^{\pi}(s,a) = E_{\pi}[G_t|S_t = s, A_t = a]$

https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc

- Dql steps