Reinforcement Learning

DQN

Александр Костин
telegramm: @Ko3tin
LinkedIn: kostinalexander

Recap: Q-Learning

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| < +\infty$

Recall the Bellman optimality equation for and apply it as an update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$$

TD-error

Bellman target

In this case, the learned action-value function, Q, directly approximates Q^*, the optimal action-value function, independent of the policy being followed.

Recap: Q-Learning

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| < +\infty$

Recall the Bellman optimality equation for and apply it as an update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$$

TD-error

Bellman target

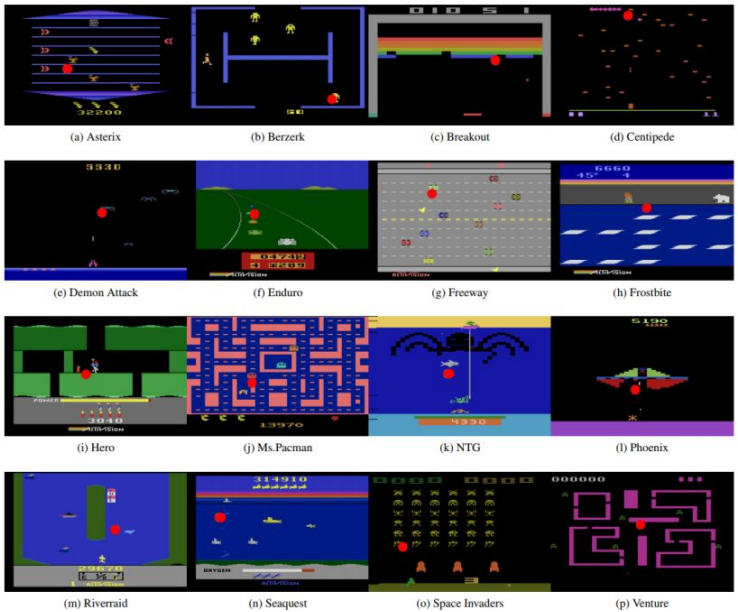
In this case, the learned action-value function, Q, directly approximates Q^*, the optimal action-value function, independent of the policy being followed.

Recall the trajectories are generated following the ε-greedy (behaviour) policy while the -function's update corresponds to the greedy (target) strategy. That is the reason why Q-learning is an off-policy method.

Atari

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| \ll +\infty$

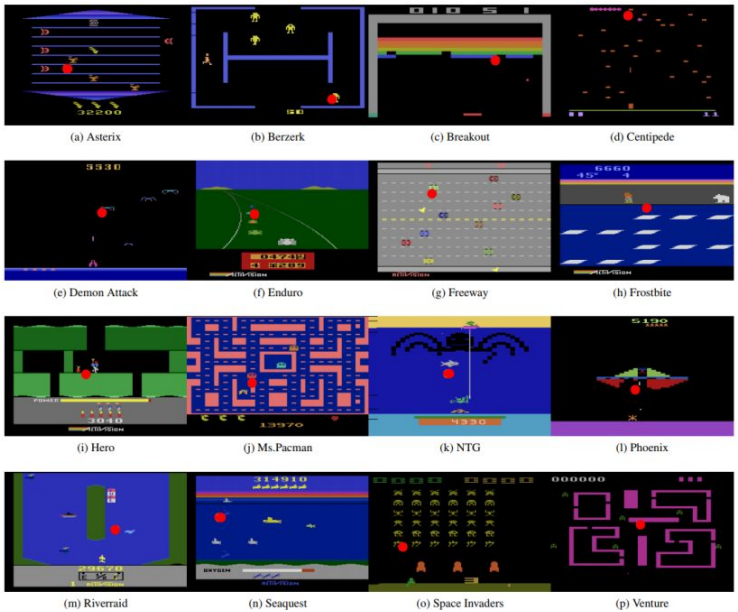
1. More than 57 different games
2. Only frames are available
3. State space is actually finite but too large to apply tabular methods
4. Few actions are available



Atari

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| \ll +\infty$

1. More than 57 different games
2. Only frames are available
3. State space is actually finite but too large to apply tabular methods
4. Few actions are available

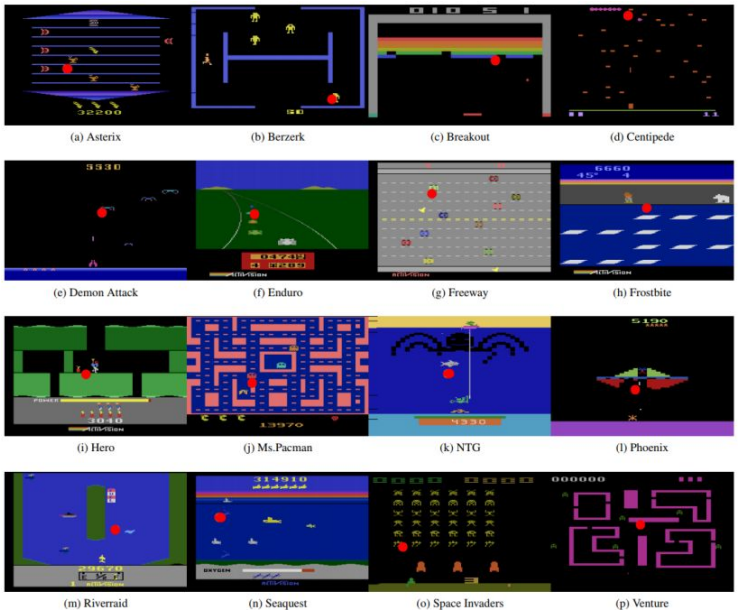


The ultimate goal is to build a universal algorithm which can be applied to all of these environments without modifications and specific hyperparameters.

Atari

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| < +\infty$

1. More than 57 different games
2. Only frames are available
3. State space is actually finite but too large to apply tabular methods
4. Few actions are available



Let’s approximate action-value function with a neural network:$Q^*(s, a) \approx Q(s, a; \theta)$

Deep Q-Networks (DQN, 2013)

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

original paper

Is Atari Environment MDP?

MDP from Frames

Preprocessing

States:

- Crop image
- Grayscale
- Downsampling
- Stack 4 consecutive frames

Environment:

- EpisodicLife
- FireReset

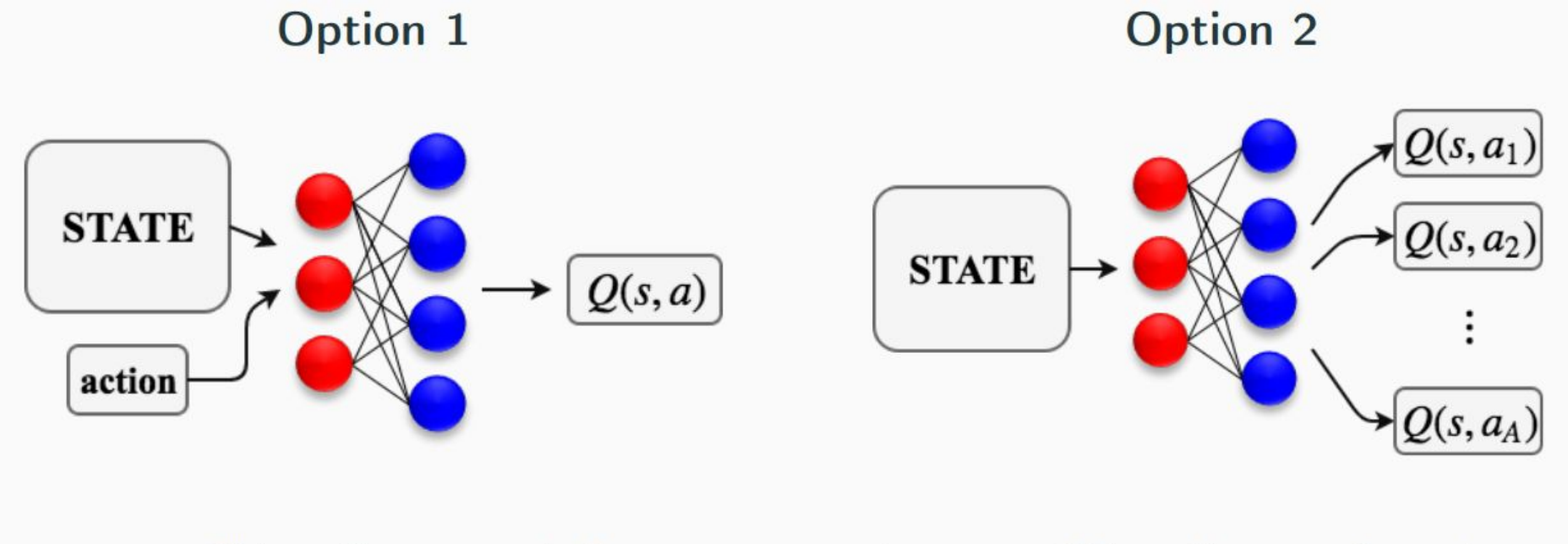
Actions' frequency:

- MaxAndSkip
- Sticky actions (we won't use)

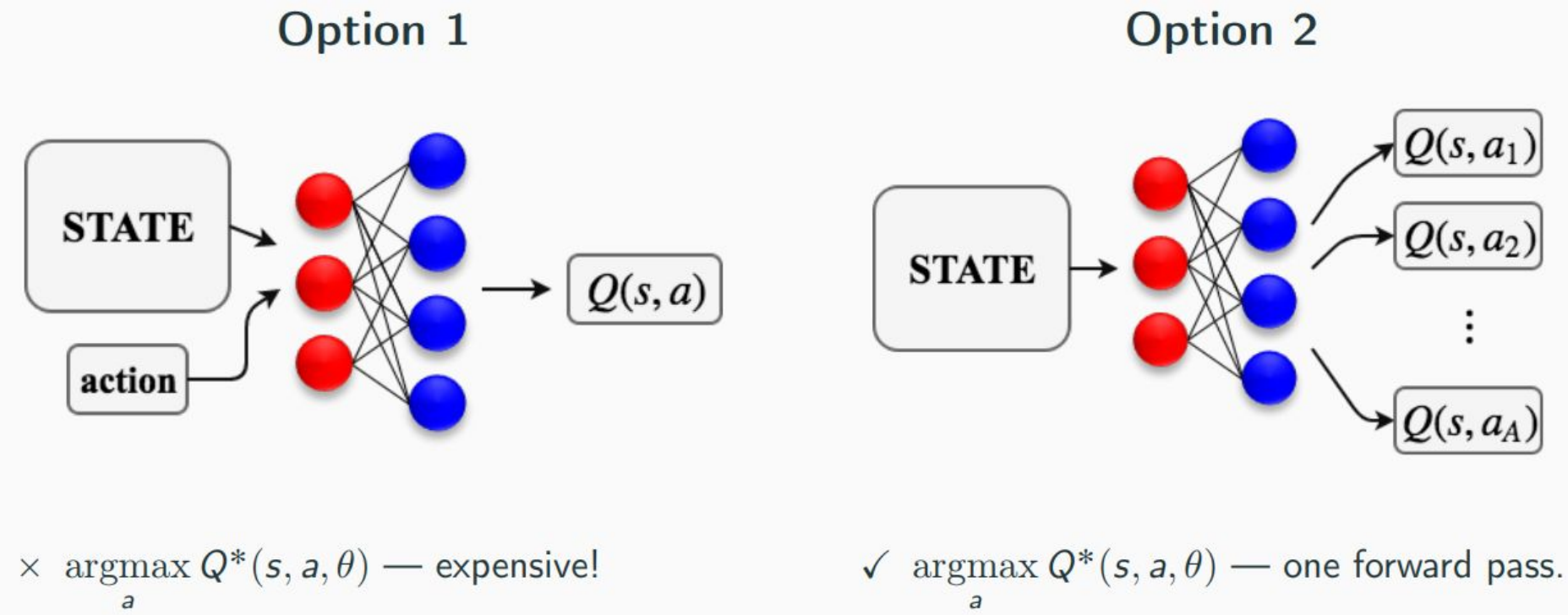
Reward:

- ClipReward ({−1,0,1})

Deep Q-network



Deep Q-network

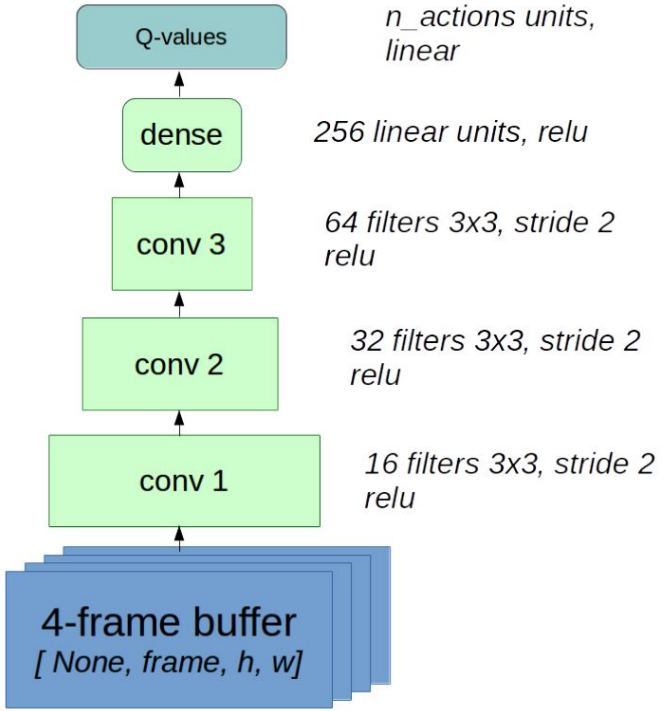


Deep Q-network

- 3-4 convolutional layers followed by 1-2 dense layers;
- stride > 1 for size reduction; linear layers still wide;

Think twice before using:

- MaxPool
- Dropout
- BatchNorm



Target

Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}[R_t + \gamma \max Q^*(S_{t+1}, a') \,|\, S_t = s, A_t = a]$$

Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$$

TD-error

Bellman target

Target

Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}[R_t + \gamma \max Q^*(S_{t+1}, a') \,|\, S_t = s, A_t = a]$$

Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$$

TD-error

Bellman target

At each iteration i:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a]$$ — target

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2]$$ — MSE loss

Gradient

At each iteration i:

$$y_i = \mathbb{E}_{s' \sim \mathscr{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a] \quad \text{— target}$$

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2] \quad \text{— MSE loss}$$

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.); s' \sim \mathscr{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

Gradient

At each iteration i:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a] \quad \text{— target}$$

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2] \quad \text{— MSE loss}$$

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.); s' \sim \mathcal{E}}\Big[\Big(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\Big) \nabla_{\theta_i} Q(s, a; \theta_i)\Big]$$

TD error

step

Bellman target

Gradient

At each iteration i:

$$y_i = \mathbb{E}_{s' \sim \mathscr{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a] \quad \text{— target}$$

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2] \quad \text{— MSE loss}$$

TD error

step

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.); s' \sim \mathscr{E}}\Big[\Big(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\Big) \nabla_{\theta_i} Q(s, a; \theta_i)\Big]$$

Bellman target

Let's apply SGD!!!

Target Network

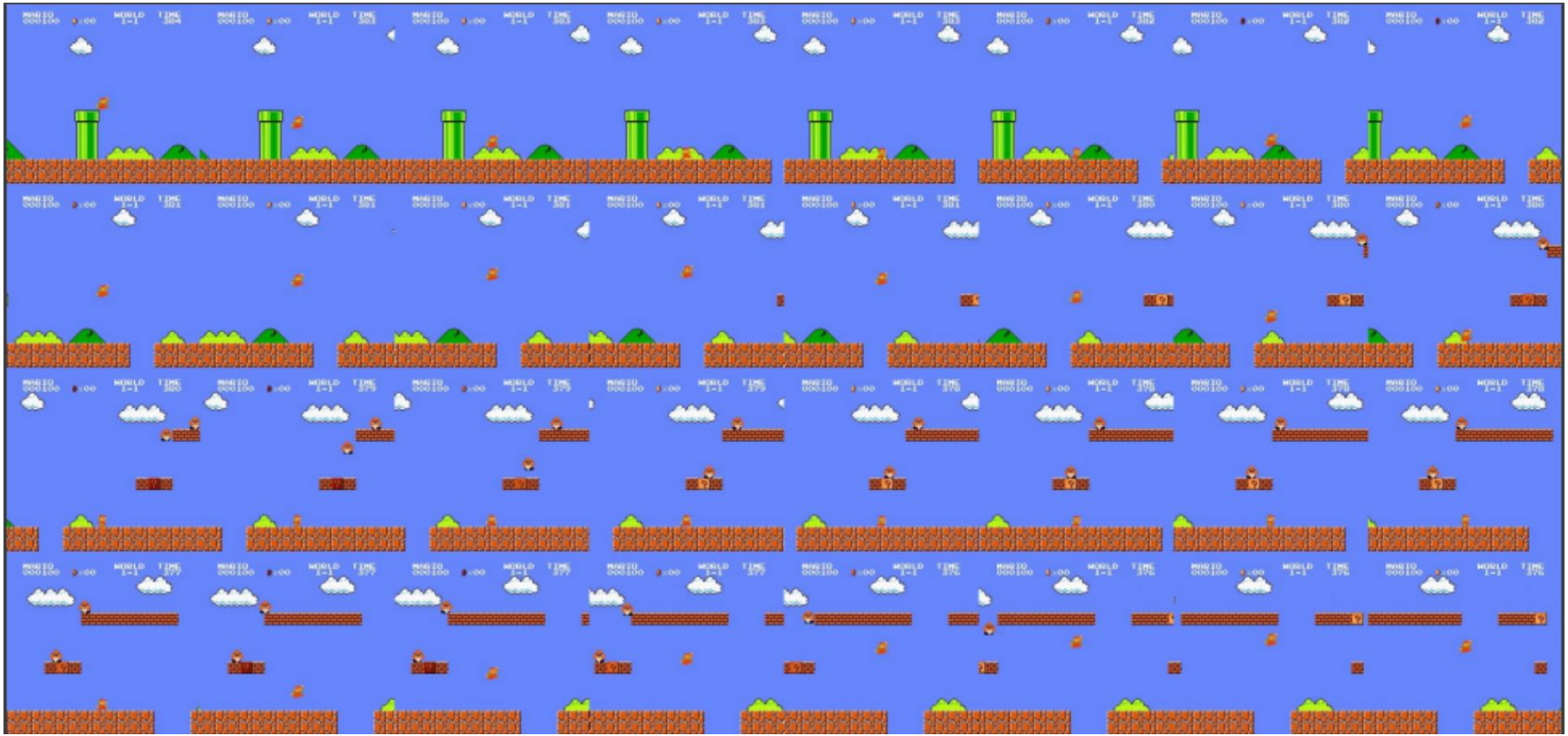
If we use weights from the previous step targets are highly non-stationary.

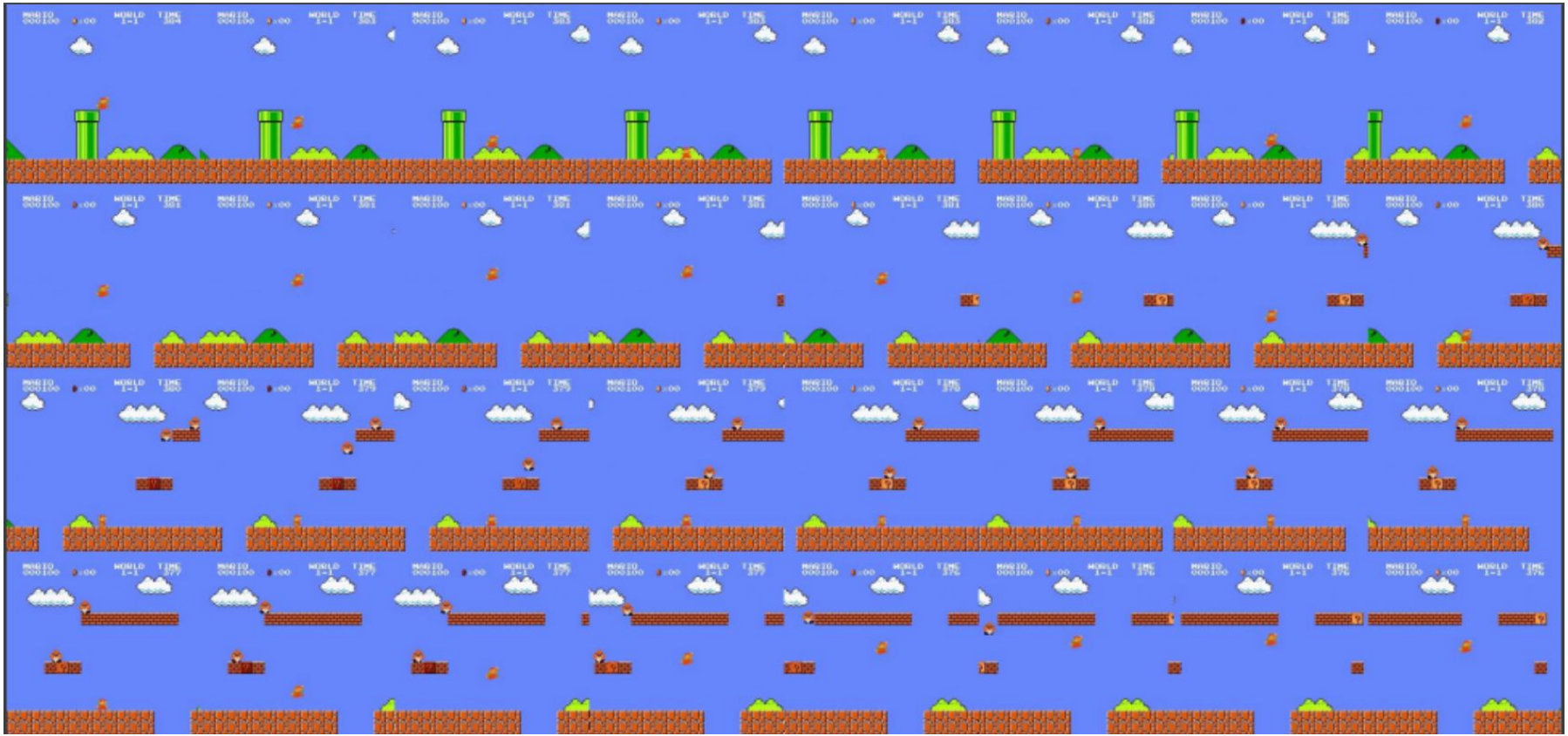
Let's update weight of the network for target generation (target network) every **K** steps or apply soft updates (Polyak update) with parameter **β**:

A) $\theta^- \leftarrow \theta$ every **K** iterations
B) $\theta^- \leftarrow (1-\beta)\theta^- + \beta\theta$ on each iteration

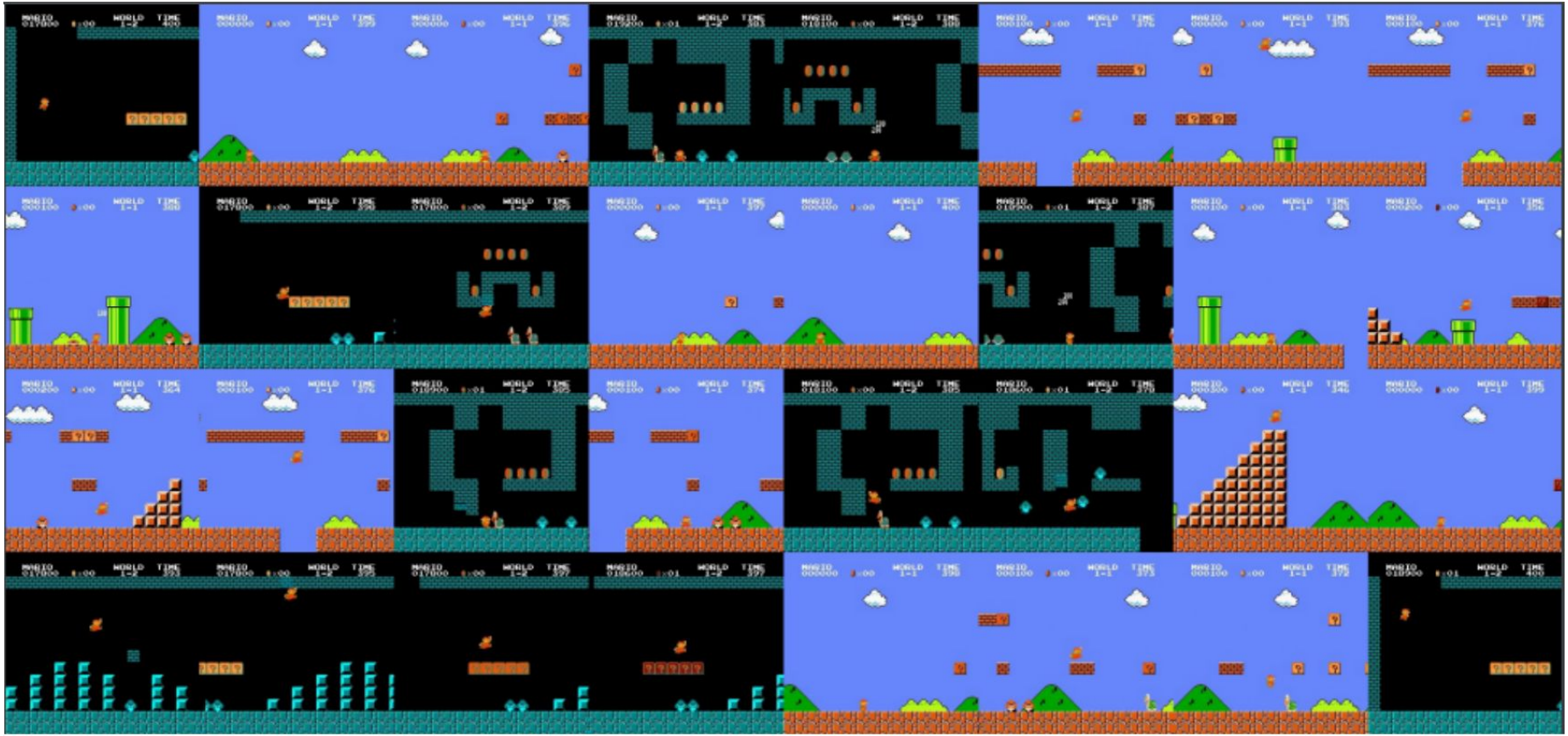
$$y_i = r + \gamma \max_{a'} Q\left(s', a', \theta^-\right)$$

$$\nabla_{\theta_i} L_i(\theta_i) = \left[r + \gamma \max_{a'} Q\left(s', a'; \theta^-\right) - Q(s, a; \theta_i)\right] \nabla_{\theta_i} Q(s, a, ; \theta_i)$$





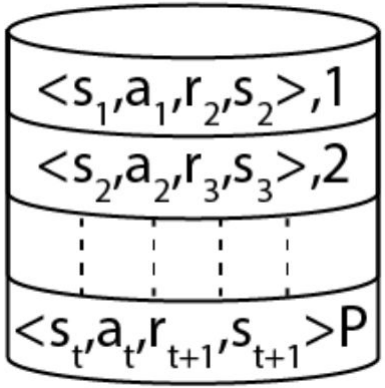
Consecutive samples are extremely correlated so batch's elements are not i.i.d.



Let’s sample randomly from the Experience Replay Buffer which stores played transitions

Experience Replay Buffer

- On each step store **<s, a, r, s', done>** in the buffer
- Sample **n** random transitions from the buffer
- Train on them



Advantages:

- No need to revisit same states many times
- Decorrelate update samples to maintain i.i.d. assumption

Disadvantages:

- Not applicable for the on-policy learning

Exploration

Recall so-called ε-greedy policy:

$$\pi(a \mid s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & a = a^\star \\ \frac{\epsilon}{|A|} & a \neq a^\star \end{cases}$$

We use ε-greedy strategies to avoid being stuck in local optima

DQN Algorithm

Deep Q-learning

Initialize $Q^*(s, a, \theta)$ arbitrarily, $\theta^- := \theta$, $\mathcal{D} = \varnothing$;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a, \theta))$;
- observe $r_k, s_{k+1}, \text{done}_{k+1}$, store $(s_k, a_k, r_k, s_{k+1}, \text{done}_{k+1})$ in $\mathcal{D}$;
- sample batch of transitions $\mathbb{T} := (s, a, r, s', \text{done})$ from $\mathcal{D}$;
- $y(\mathbb{T}) := r + \gamma(1 - \text{done}) \max_{a'} Q^*(s', a', \theta^-)$;
- perform a step of gradient descent:

$$\theta \leftarrow \theta - \frac{\alpha}{B} \sum_{\mathbb{T}} \nabla_\theta \left(Q^*(s, a, \theta) - y(\mathbb{T})\right)^2$$

- update target network: if $k \bmod K = 0$: $\theta^- \leftarrow \theta$

Rainbow (2017)

Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel DeepMind
Joseph Modayil DeepMind
Hado van Hasselt DeepMind
Tom Schaul DeepMind
Georg Ostrovski DeepMind

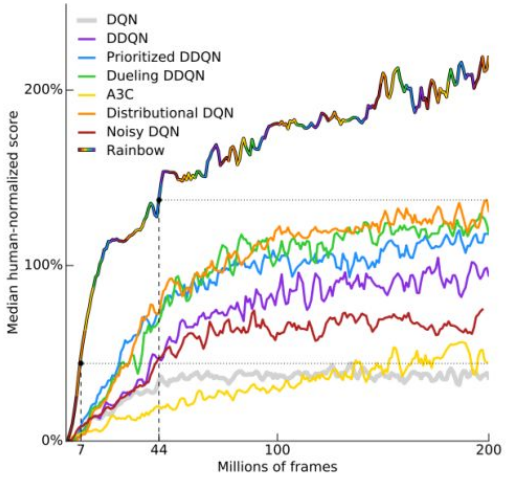
Will Dabney DeepMind
Dan Horgan DeepMind
Bilal Piot DeepMind
Mohammad Azar DeepMind
David Silver DeepMind

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 exten



original paper

Overestimation Bias

The overestimation of the Q-function by the algorithms which can be caused by several reasons:

1. Maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to a significant positive bias due to approximation error.
2. Due to using the same samples (plays) both to determine the maximizing action and to estimate its value.

Action selection

$$\max_{a'} Q(s', a') = Q(s', \mathrm{argmax}_{a'} Q(s', a'))$$

Action evaluation

Double DQN

Let's decouple action selection and action evaluation. Like in the tabular setting we can use two weakly correlated networks with independent buffers D_i:

$$y_1 = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_1); \theta_2)$$

$$y_2 = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_2); \theta_1)$$

but it can be too expensive…

Double DQN

Let's decouple action selection and action evaluation. Like in the tabular setting we can use two weakly correlated networks with independent buffers D_i:

$$y_1 = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_1); \theta_2)$$

$$y_2 = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_2); \theta_1)$$

but it can be too expensive…

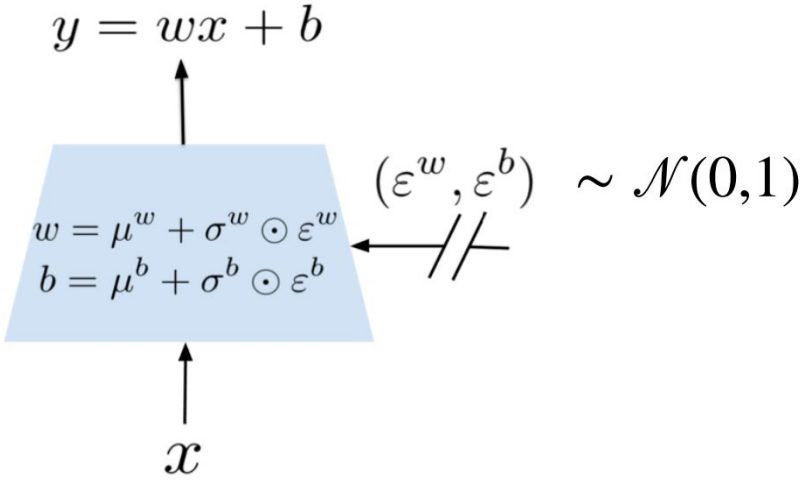
We can use the target network as the second network:

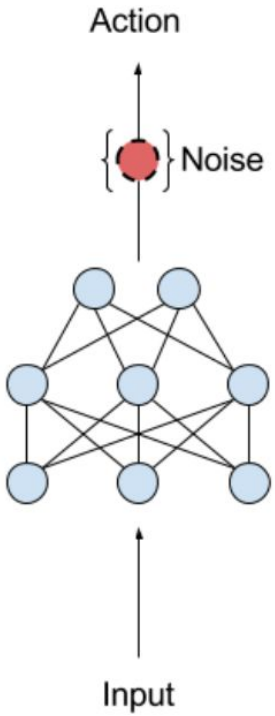
$$y = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_t); \theta_t^-)$$

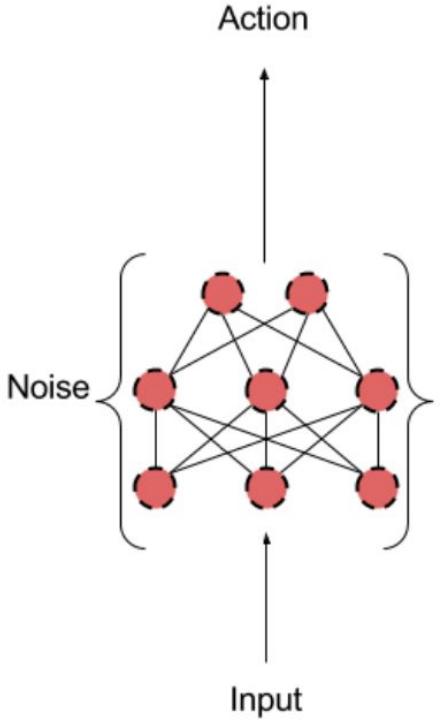
Noisy Networks

Issues:

- State-independent exploration
- ε-greedy is naive



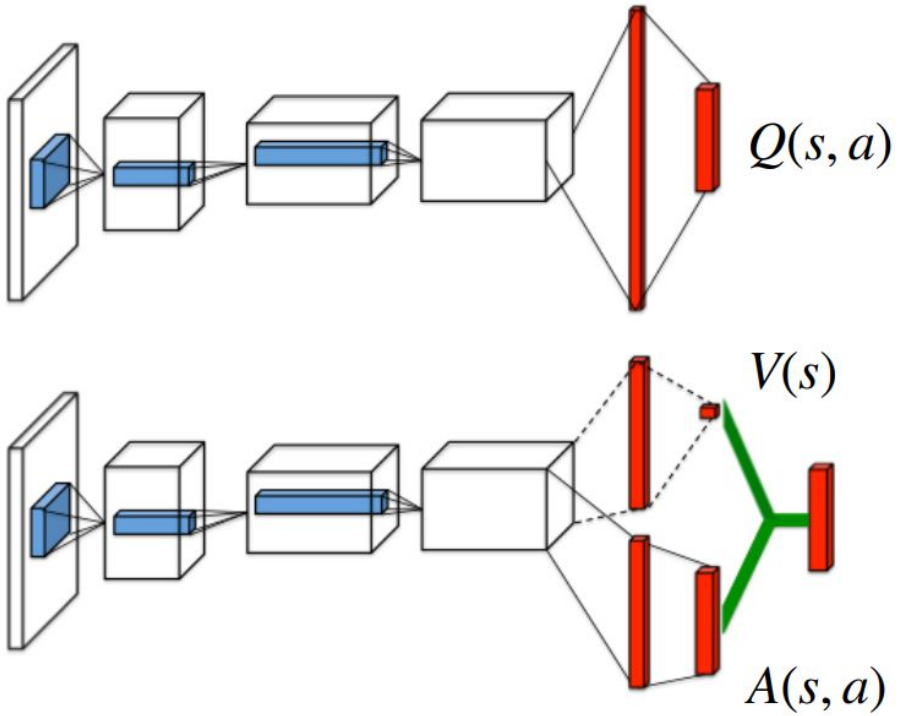




Dueling DQN

A(s,a)=Q(s,a)-V(s) — is a relative measure of importance of each action, advantage.

$$Q(s,a) = V(s) + A(s,a)$$



Dueling DQN

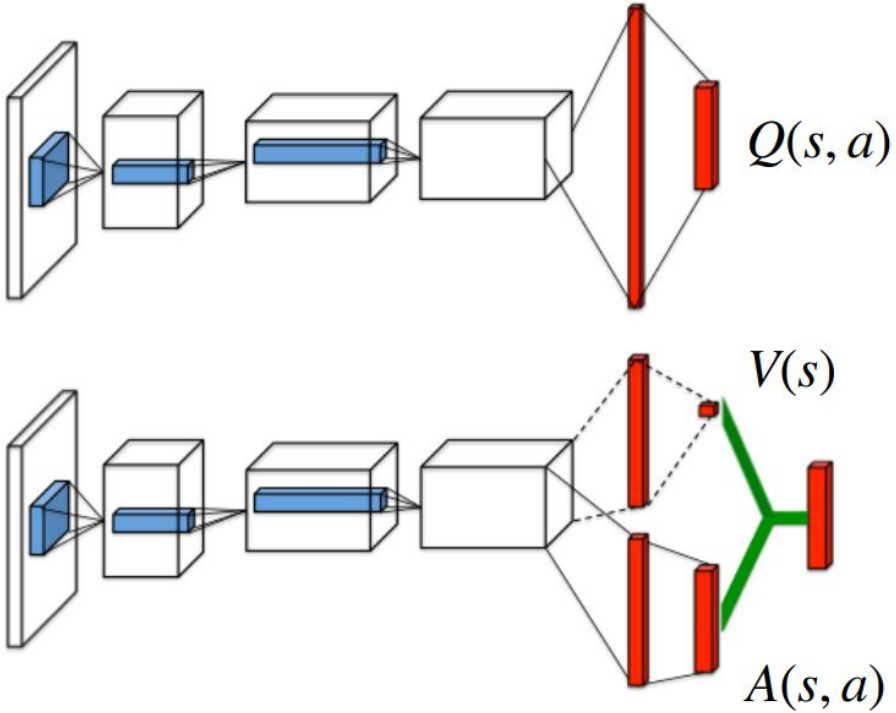
A(s,a)=Q(s,a)-V(s) — is a relative measure of importance of each action, advantage.

~~$Q(s,a) = V(s) + A(s,a)$~~

$$Q(s,a) = V(s) + \left(A(s,a) - \max_a A(s,a)\right)$$

$$since\ V(s) = \max_a Q(s,a)$$

$$or\ Q(s,a) = V(s) + (A(s,a) - mean_a A(s,a))$$



Multi-step DQN

Problem: delayed rewards + compounding error

$$y = r + \gamma r' + \gamma r'' + \ldots + \gamma^n \max_{a^{(n)}} Q(s^{(n)}, a^{(n)})$$

We assume that all transitions are generated with greedy policy but it's not the case.

Important!

- ✓ can *significantly* help with this issue;
- × theoretically can't be done off-policy;

Prioritized Experience Replay

The uniform sampling from the experience replay make an agent replay transitions at the same frequency that were originally experienced, regardless of their significance.

Prioritized Experience Replay

The uniform sampling from the experience replay make an agent replay transitions at the same frequency that were originally experienced, regardless of their significance.

$\delta_i = y_i - Q(s, a; \theta) \; - \; TD \; error$

$p_i = |\delta_i| + \epsilon, \; \epsilon > 0$

$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ — is a probability of sampling transition i

Prioritized Experience Replay

The uniform sampling from the experience replay make an agent replay transitions at the same frequency that were originally experienced, regardless of their significance.

$\delta_i = y_i - Q(s, a; \theta) \; - \; TD \; error$

$p_i = |\delta_i| + \epsilon, \; \epsilon > 0$

$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ — is a probability of sampling transition i

However we induce a bias in a Q-function approximation!

Prioritized Experience Replay

The uniform sampling from the experience replay make an agent replay transitions at the same frequency that were originally experienced, regardless of their significance.

$\delta_i = y_i - Q(s, a; \theta) \; - \; TD \; error$

$p_i = |\delta_i| + \epsilon, \; \epsilon > 0$

$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ — is a probability of sampling transition i

Let's use importance sampling (IS):

$$Loss \approx E_{i \sim P(i)} \left(\frac{1}{P(i)} \right)^{\beta(t)} \delta_i, \;\; where \, \beta \, anneals \, from \, 0 \, to \, 1.$$

Ablation Study

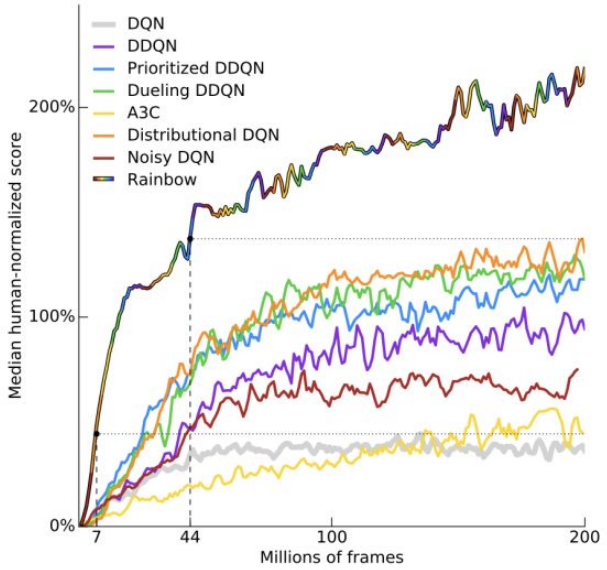


Figure 1: **Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final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

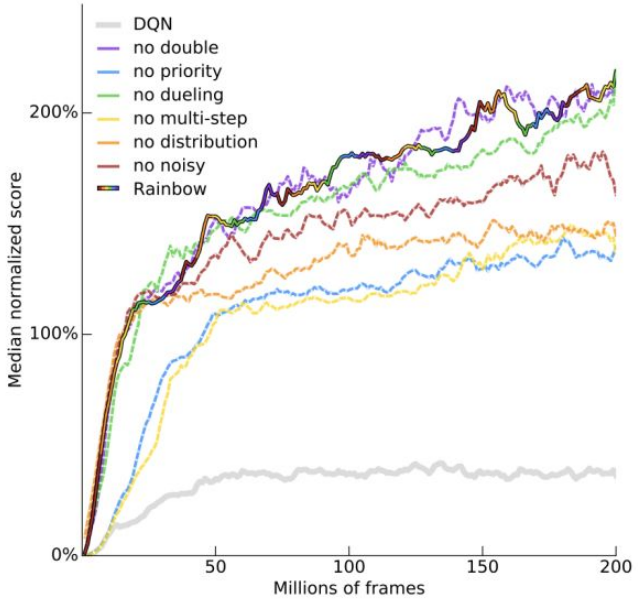


Figure 3: **Median human-normalized performance** across 57 Atari games, as a function of time. We compare our integrated agent (rainbow-colored) to DQN (gray) and to six different ablations (dashed lines). Curves are smoothed with a moving average over 5 points.

Recap: MDP

MDP is a 4-tuple **(S, A, p, r)**

- **S** is a state space
- **A** is an action space
- **p = p(s' | s, a)** is a state-transition function
- **r = AxS→R** is a reward function giving an expected reward: **r(s,a)=E[r|s,a]**

Recap: MDP

MDP is a 4-tuple **(S, A, p, r)**

- **S** is a state space
- **A** is an action space
- **p = p(s' | s, a)** is a state-transition function
- **r = AxS→R** is a reward function giving an expected reward: **r(s,a)=E[r|s,a]**

POMDP

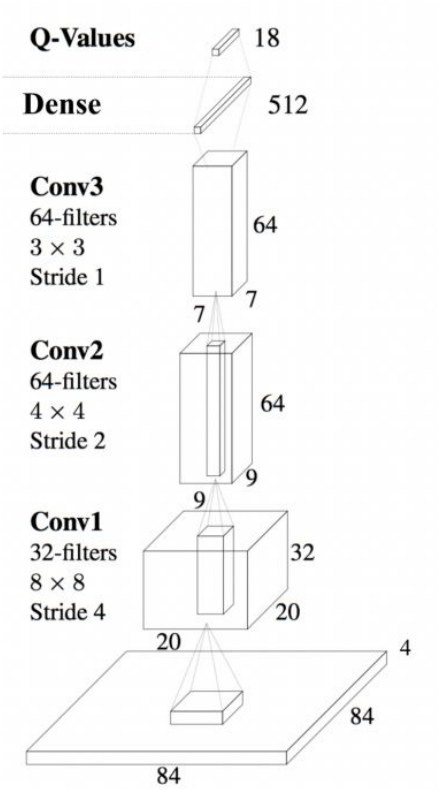
- **(S, A, r)** are the same as in MDP
- **O** is a set of possible observations
- **p = p(s' | s, a)** is a state-transition function
- **p = p(o | s', a) = $P(o_t=o|s_{t+1}=s',a_t=a)$** is a observation-transition function

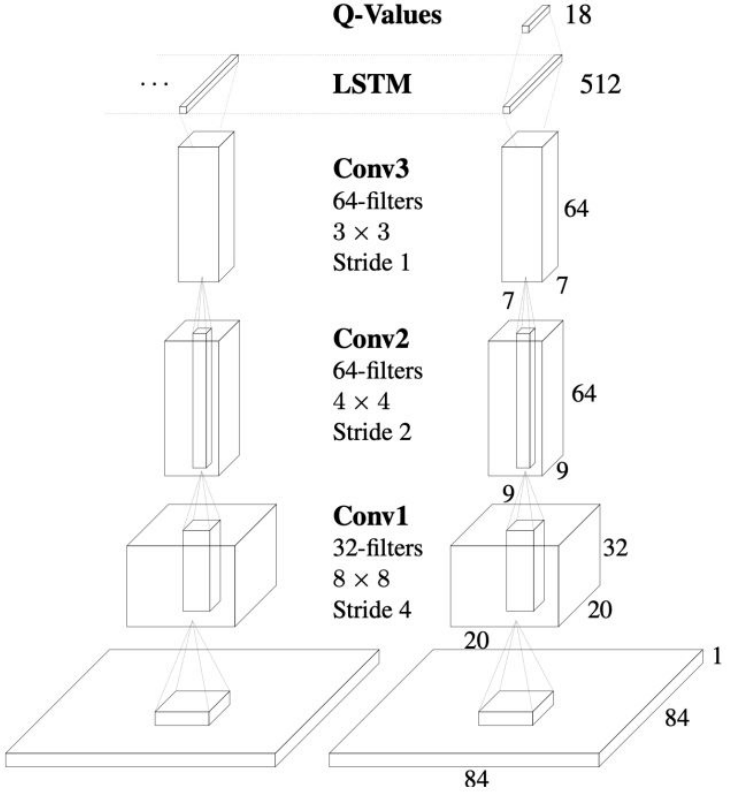
DRQN

Vanilla Deep Q-Learning has no explicit mechanisms for deciphering the underlying state of the POMDP and is only effective if the observations are reflective of underlying system states. In the general case, estimating a Q-value from an observation can be arbitrarily bad since **$Q(o, a; \theta) \neq Q(s, a; \theta)$**

- Let's equip agent with memory **h**
- **$Q(s_t, a_t) \approx Q(o_t, h_{t-1}, a_t)$**
- **$h_t = LSTM(o_t, h_{t-1})$**

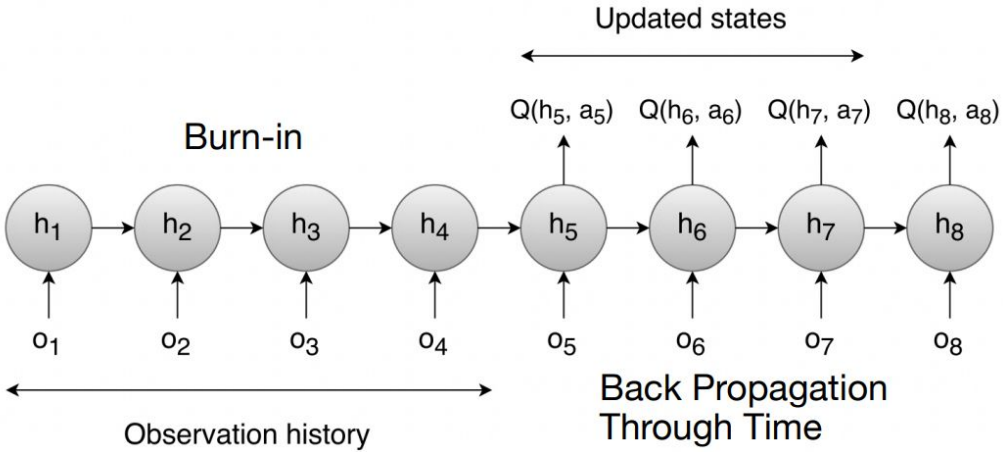
DQN vs DRQN

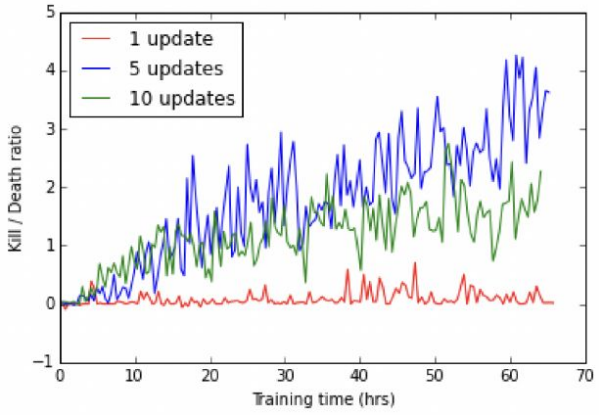

Q-Values
18
Dense
512
Conv3
64-filters
3 × 3
Stride 1
64
7
7
Conv2
64-filters
4 × 4
Stride 2
64
9
9
Conv1
32-filters
8 × 8
Stride 4
32
20
20
4
84
84


Q-Values
18
...
LSTM
512
Conv3
64-filters
3 × 3
Stride 1
64
7
7
Conv2
64-filters
4 × 4
Stride 2
64
9
9
Conv1
32-filters
8 × 8
Stride 4
32
20
20
1
84
84

DRQN Experience Replay

- Sample random time step
- Consider **N** consecutive transitions
- Update only last **M**





Goal-conditioned RL

G - task to complete

r = r(s, a | G) - depends on goal

How to train for multiple goals?

Possible tasks:

1) grab an apple
2) grab a pear

Goal-conditioned RL

G - task to complete

r = r(s, a | G) - depends on goal

How to train for multiple goals?

One could train an agent for every possible goal.

But can we do better?

Possible tasks:

1) grab an apple
2) grab a pear

Hindsight Experience Replay

The idea is simple:

1) experience some episode {s,a,r,s,a,r,...}
2) store it in the buffer
3) use another goal when train on it (like if agent correctly completed its task)

original paper