Using ensembles for reducing the estimation bias in Q-Learning algorithms

Oren Peer

Advisor: Prof. Ron Meir

June 2021

TECHNION
Israel Institute
of Technology

Reinforcement Learning
Research Labs

Based on our recently accepted paper:

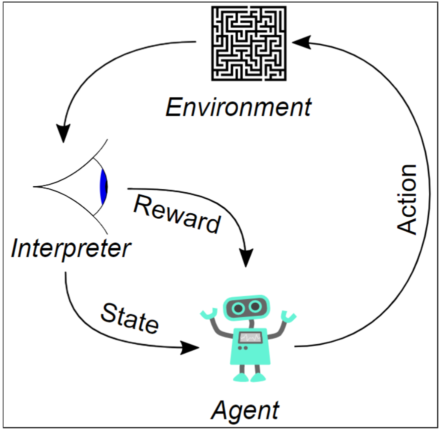
Ensemble Bootstrapping for Q-Learning

Oren Peer, Chen Tessler, Nadav Merlis, Ron Meir

The Thirty-eighth International Conference on Machine Learning (ICML 2021)

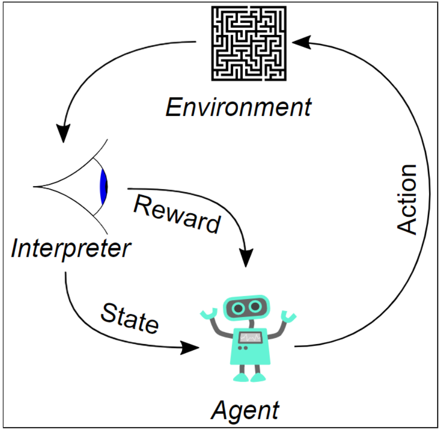
Thanks,

Reinforcement learning (RL)



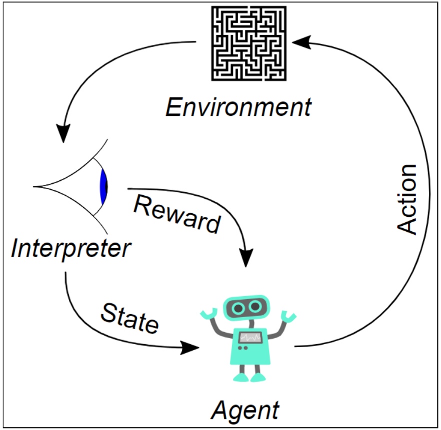
Reinforcement learning (RL)

- State – $s_t \in \mathcal{S}$.
  Action – $a_t \in A$.
  Scalar reward – r_t.
  Transition probability – $P(s_{t+1}|s_t, a)$.
  Discount factor $\gamma \in [0, 1]$.



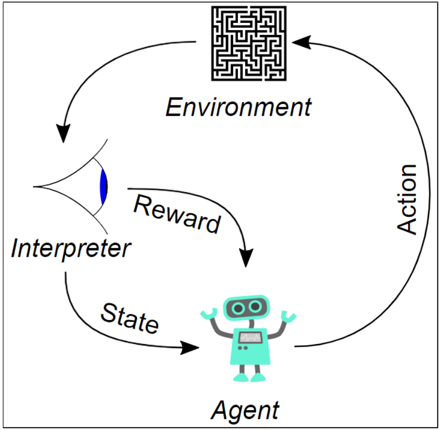
Reinforcement learning (RL)

- State – $s_t \in \mathcal{S}$.
  Action – $a_t \in A$.
  Scalar reward – r_t.
  Transition probability – $P(s_{t+1}|s_t, a)$.
  Discount factor $\gamma \in [0, 1]$.
- Policy – $\pi : \mathcal{S} \to \mathcal{A}$, how to act in a given state.



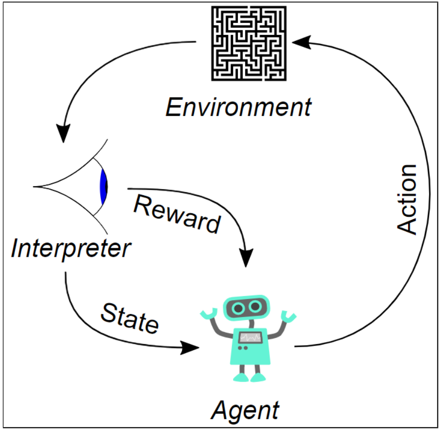
Reinforcement learning (RL)

- State – $s_t \in \mathcal{S}$.
  Action – $a_t \in A$.
  Scalar reward – r_t.
  Transition probability – $P(s_{t+1}|s_t, a)$.
  Discount factor $\gamma \in [0, 1]$.
- Policy – $\pi : \mathcal{S} \to \mathcal{A}$, how to act in a given state.
- Value – $v^\pi(s) = \mathbb{E}^\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)|s_0 = s\right]$.



Reinforcement learning (RL)

- State – $s_t \in \mathcal{S}$.
  Action – $a_t \in A$.
  Scalar reward – r_t.
  Transition probability – $P(s_{t+1}|s_t, a)$.
  Discount factor $\gamma \in [0, 1]$.
- Policy – $\pi : \mathcal{S} \to \mathcal{A}$, how to act in a given state.
- Value – $v^\pi(s) = \mathbb{E}^\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | s_0 = s\right]$.
- Q-function – $Q^\pi(s, a) = \mathbb{E}^\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) | s_0 = s, a_0 = a\right]$



Motivation

- Value based methods: learn the *value* of each state $\rightarrow$ Infer optimal policy.

Motivation

- Value based methods: learn the *value* of each state $\rightarrow$ Infer optimal policy.
    - Temporal Difference (TD) methods - *Bootstrapping* using the next state(s) values.

Motivation

- Value based methods: learn the *value* of each state → Infer optimal policy.
  - Temporal Difference (TD) methods - *Bootstrapping* using the next state(s) values.
    - Tabular methods - Q-Learning, SARSA...

Motivation

- Value based methods: learn the *value* of each state → Infer optimal policy.
  - Temporal Difference (TD) methods - *Bootstrapping* using the next state(s) values.
    - Tabular methods - Q-Learning, SARSA...
    - Large state spaces and large/continues state/action spaces,
      ↳ Function approximations (e.g., Deep Q-Networks- DQN).

Motivation

- Value based methods: learn the *value* of each state → Infer optimal policy.
  - Temporal Difference (TD) methods - *Bootstrapping* using the next state(s) values.
    - Tabular methods - Q-Learning, SARSA...
    - Large state spaces and large/continues state/action spaces,
      ↳ Function approximations (e.g., Deep Q-Networks- DQN).

Bootstrapping using the *Optimal Bellman Operator* causes TD algorithms to overestimate the value function → Slow convergence!☹

Q-Learning (QL)– Watkins and Dayan (1992)

- QL learns Q^* from transition tuples (s, a, r, s'), by iteratively applying the optimal Bellman operator:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a')).$$

Q-Learning (QL)– Watkins and Dayan (1992)

- QL learns Q^* from transition tuples (s, a, r, s'), by iteratively applying the optimal Bellman operator:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a')).$$

  - The *max* term causes QL to *overestimate* the next-state Q-values.
    $\rightarrow$ slow convergence and poor performance[1].

[1] Van Hasselt 2010.

Q-Learning (QL)– Watkins and Dayan (1992)

- QL learns Q^* from transition tuples (s, a, r, s'), by iteratively applying the optimal Bellman operator:

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha\left(r_t + \gamma \max_{a'} Q(s', a')\right).$$

  - The *max* term causes QL to *overestimate* the next-state Q-values.
    $\rightarrow$ slow convergence and poor performance[1].
  - Overestimation - main reason to the divergence of QL-based algorithms[2].

[1] Van Hasselt 2010.
[2] Van Hasselt et al. 2018.

Example - Chain MDP

Consider the following Chain-MDP where $\mu_i < 0$.

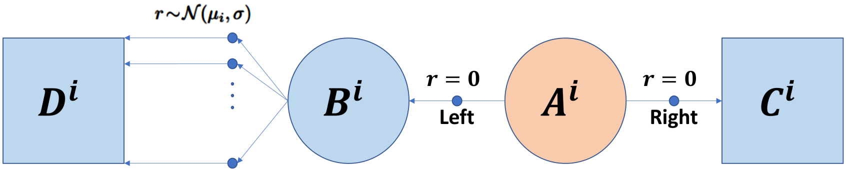
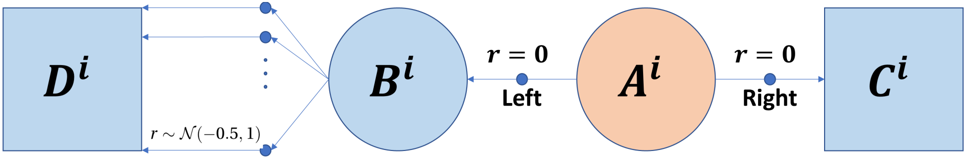


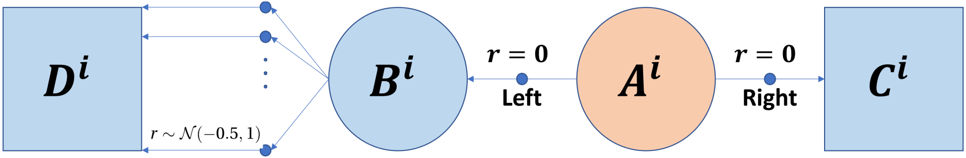
Figure: The Chain MDP

Example - Chain MDP



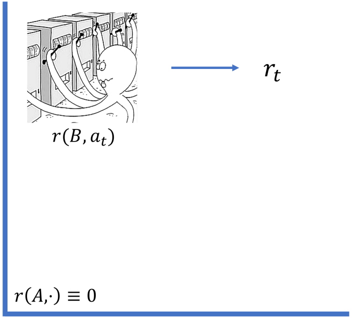
Optimal policy:
Pick "Right" action from A^i. Forever!

Example - Chain MDP

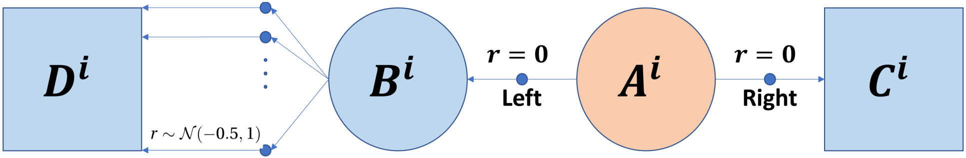


$Q(B,\cdot)$	
$Q(B,a_1)$	0
$Q(B,a_2)$	0
$Q(B,a_3)$	0
$Q(B,a_4)$	0
$Q(B,a_5)$	0
$Q(B,a_6)$	0

$Q(A,\cdot)$	
$Q(A,Left)$	0
$Q(A,Right)$	0

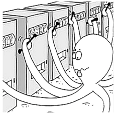


Update Rule: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a'))$



$Q(B,\cdot)$	
$Q(B,a_1)$	-0.07
$Q(B,a_2)$	$\boxed{0}$
$Q(B,a_3)$	0
$Q(B,a_4)$	0
$Q(B,a_5)$	0
$Q(B,a_6)$	0

$Q(A,\cdot)$	
$Q(A,Left)$	-0.04
$Q(A,Right)$	$\boxed{0}$



$\longrightarrow r_t$

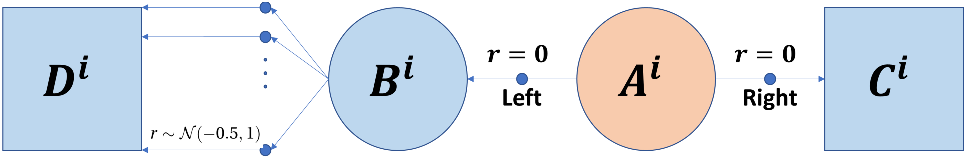
$r(B,a_t)$

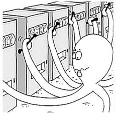
$$\boxed{\begin{gathered} a_t = a_1 \\ r(B,a_1) = -0.1 \end{gathered}}$$

$r(A,\cdot) \equiv 0$

Update Rule: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a'))$

Example - Chain MDP





r_t

$r(B, a_t)$

$a_t = a_2$
$r(B, a_1) = 0.2$

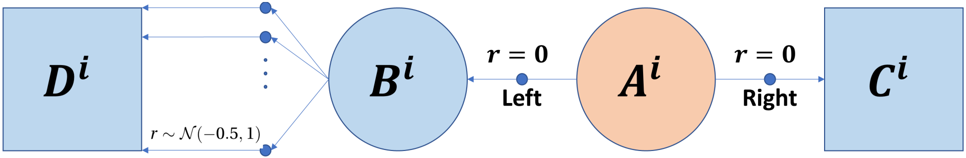
$r(A, \cdot) \equiv 0$

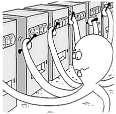
$Q(B,\cdot)$	
$Q(B, a_1)$	-0.07
$Q(B, a_2)$	$\boxed{0.15}$
$Q(B, a_3)$	0
$Q(B, a_4)$	0
$Q(B, a_5)$	0
$Q(B, a_6)$	0

$Q(A,\cdot)$	
$Q(A, Left)$	-0.01
$Q(A, Right)$	$\boxed{0}$

Update Rule: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a'))$

Example - Chain MDP





r_t

$r(B, a_t)$

$a_t = a_2$
$r(B, a_2) = -0.1$

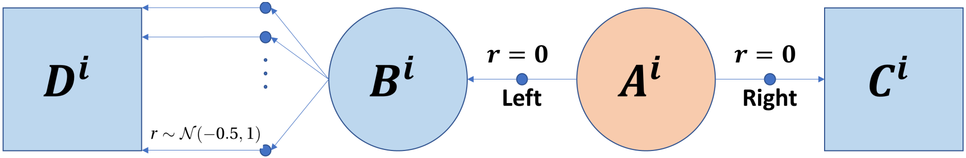
$r(A, \cdot) \equiv 0$

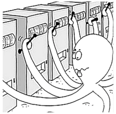
$Q(B,\cdot)$	
$Q(B,a_1)$	-0.7
$Q(B,a_2)$	$\boxed{0.05}$
$Q(B,a_3)$	0
$Q(B,a_4)$	0
$Q(B,a_5)$	0
$Q(B,a_6)$	0

$Q(A,\cdot)$	
$Q(A, Left)$	$\boxed{0.02}$
$Q(A, Right)$	0

Update Rule: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a'))$

Example - Chain MDP





$\rightarrow r_t$

$r(B, a_t)$

$a_t = a_2$
$r(B, a_2) = -0.51$

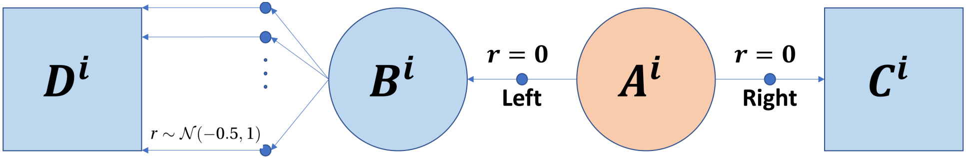
$r(A, \cdot) \equiv 0$

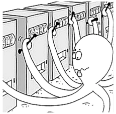
$Q(B, \cdot)$	
$Q(B, a_1)$	-0.7
$Q(B, a_2)$	-0.34
$Q(B, a_3)$	$\boxed{0}$
$Q(B, a_4)$	0
$Q(B, a_5)$	0
$Q(B, a_6)$	0

$Q(A, \cdot)$	
$Q(A, Left)$	$\boxed{0.01}$
$Q(A, Right)$	0

Update Rule: $Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha (r_t + \gamma \max_{a'} Q(s', a'))$

Example - Chain MDP





$r(B, a_t)$

$\longrightarrow r_t$

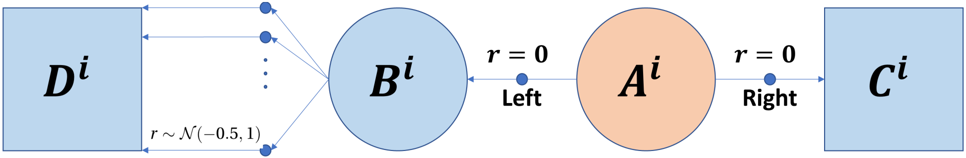
$a_t = a_3$
$r(B, a_2) = -0.44$

$r(A,\cdot) \equiv 0$

$Q(B,\cdot)$	
$Q(B, a_1)$	-0.7
$Q(B, a_2)$	-0.34
$Q(B, a_3)$	-0.32
$Q(B, a_4)$	$\boxed{0}$
$Q(B, a_5)$	0
$Q(B, a_6)$	0

$Q(A,\cdot)$	
$Q(A, Left)$	$\boxed{0.005}$
$Q(A, Right)$	0

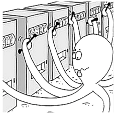
Update Rule: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a'))$



$Q(B,\cdot)$	
$Q(B,a_1)$	-0.81
$Q(B,a_2)$	-0.55
$Q(B,a_3)$	-0.32
$Q(B,a_4)$	-0.3
$Q(B,a_5)$	-0.6
$Q(B,a_6)$	$\boxed{-0.2}$

$Q(A,\cdot)$	
$Q(A,Left)$	-0.15
$Q(A,Right)$	$\boxed{0}$

That took some time...



$\longrightarrow r_t$

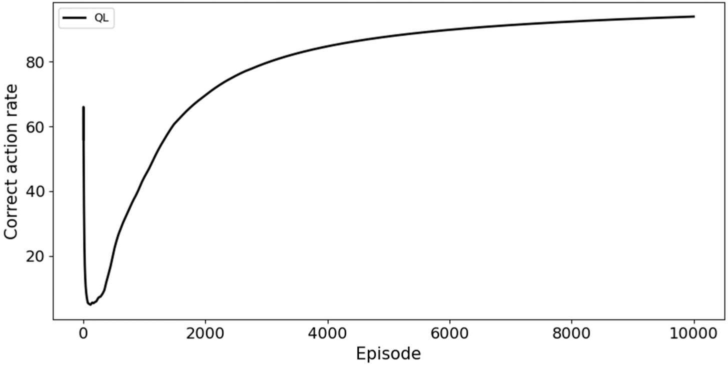
$r(B,a_t)$

$a_t = a_6$
$r(B,a_2) = -0.3$

$r(A,\cdot) \equiv 0$

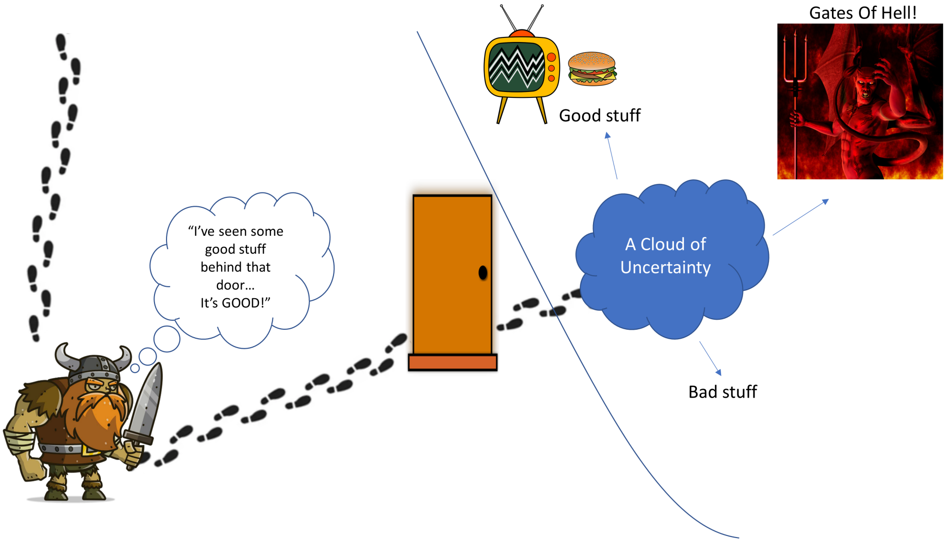
Update Rule: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a'))$

Slow Convergence



Slow convergence of QL over the chain MDP, available actions at state $B - 10$. $r(B, \cdot) \sim \mathcal{N}(-0.2, 1)$

General Optimism



QL Overestimation

- QL overestimates the Q-function. This can be bad **sometimes**.

QL Overestimation

- QL overestimates the Q-function. This can be bad **sometimes**.
- The problem gets worse when dealing with function approximators:

QL Overestimation

- QL overestimates the Q-function. This can be bad **sometimes**.
- The problem gets worse when dealing with function approximators:
  - Target Approximation Error (TAE) [Thrun and Schwartz 1993].

QL Overestimation

- QL overestimates the Q-function. This can be bad **sometimes**.
- The problem gets worse when dealing with function approximators:
  - Target Approximation Error (TAE) [Thrun and Schwartz 1993].
  - **Neural Networks: DeepRL -** 'The Deadly Triad' [Van Hasselt et al. 2018].

Double Q-Learning (DQL)– Van Hasselt (2010)

- Avoiding overestimation, DQL combines two Q-estimators: Q^A and Q^B, each updated over a **unique subset** of gathered experience.

Double Q-Learning (DQL)– Van Hasselt (2010)

- Avoiding overestimation, DQL combines two Q-estimators: Q^A and Q^B, each updated over a **unique subset** of gathered experience.

- Two-Phased update step of estimator A:

$$
\begin{aligned}
(1) \qquad & \hat{a}^*_A = \operatorname*{argmax}_{a'} Q^A(s_{t+1}, a') \\
(2) \qquad & Q^A(s_t, a_t) \leftarrow (1 - \alpha_t) Q^A(s_t, a_t) \\
& \qquad\qquad + \alpha_t \left(r_t + \gamma Q^B (s_{t+1}, \hat{a}^*_A) \right)
\end{aligned}
$$

Double Q-Learning (DQL)– Van Hasselt (2010)

- Avoiding overestimation, DQL combines two Q-estimators: Q^A and Q^B, each updated over a **unique subset** of gathered experience.

- Two-Phased update step of estimator A:

$$
\begin{aligned}
&(1) \qquad \hat{a}_A^* = \underset{a'}{\operatorname{argmax}}\, Q^A(s_{t+1}, a') \\
&(2) \qquad Q^A(s_t, a_t) \leftarrow (1-\alpha_t) Q^A(s_t, a_t) \\
&\qquad\qquad\qquad + \alpha_t \left(r_t + \gamma Q^B(s_{t+1}, \hat{a}_A^*) \right)
\end{aligned}
$$

For updating estimator B, the indices A and B are flipped.

Double Q-Learning (DQL)– Van Hasselt (2010)

DQL combines two Q-estimators: Q^A and Q^B, each updated over a unique subset of gathered experience. $UPDATE\_A \sim Bern(p = 0.5)$.

Algorithm 1: Double Q-Learning (DQL)

Initialize: Two Q-tables: Q^A and Q^B , s_0.

for $t = 1, \ldots, T$ **do**

  Choose action $a_t = \arg\max_a \left[Q^A(s_t, a) + Q^B(s_t, a)\right]$

  $a_t = \text{explore}(a_t)$; // e.g. ϵ-greedy

  $s_{t+1}, r_t \leftarrow \text{env.step}(s_t, a_t)$

  if *UPDATE_A.Sample() = 1* **then**

    Define $a^* = \arg\max_a Q^A(s_{t+1}, a)$

    $Q^A(s_t, a_t) \leftarrow Q^A(s_t, a_t) + \alpha_t \left(r_t + \gamma Q^B(s_{t+1}, a^*) - Q^A(s_t, a_t)\right)$

  else

    // UPDATE_B

    Define $b^* = \arg\max_a Q^B(s_{t+1}, a)$

    $Q^B(s_t, a_t) \leftarrow Q^B(s_t, a_t) + \alpha_t \left(r_t + \gamma Q^A(s_{t+1}, b^*) - Q^B(s_t, a_t)\right)$

Result: $\{Q^A, Q^B\}$

Double Q-Learning (DQL)– Van Hasselt (2010)

- DQL mitigates the overestimation. But, results with underestimation.

Example - Chain MDP

Consider the following Chain-MDP where $\mu_i > 0$

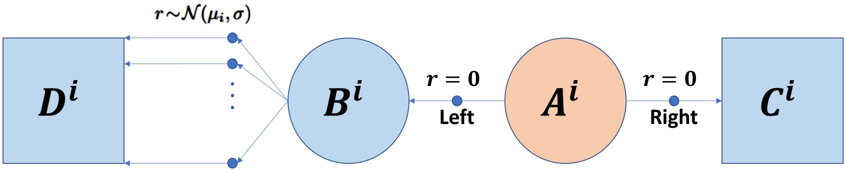
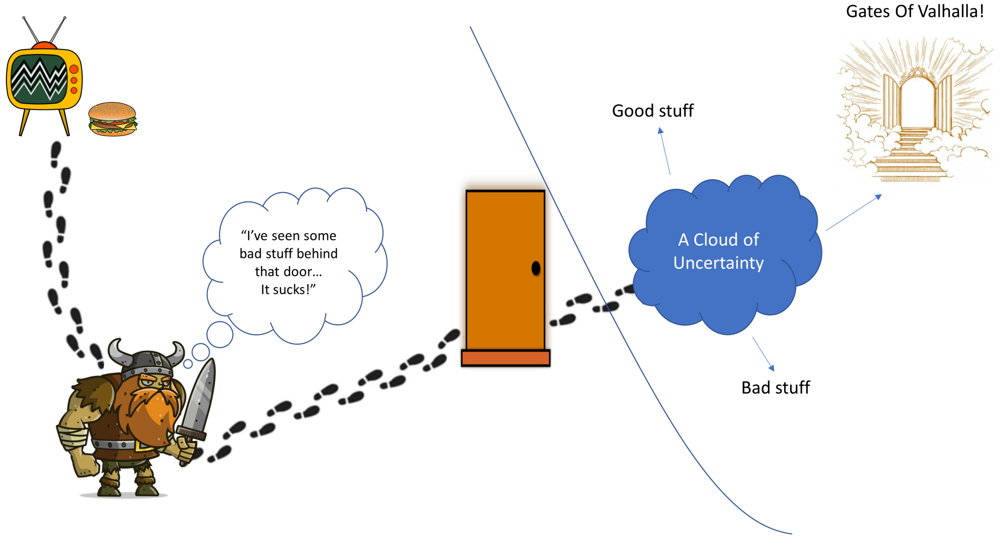


Figure: The Chain MDP

General Pessimism



QL, DQL and the 'nature of the environment'

- QL overestimates $\rightarrow$ **'optimistic'**. Hence, performs well in environments where **optimism is beneficial**!

QL, DQL and the ‘nature of the environment’

- QL overestimates → **‘optimistic’**. Hence, performs well in environments where **optimism is beneficial**!
- DQL underestimates → **‘pessimistic’**. Hence, performs well in environments where **pessimism is beneficial**!

QL, DQL and the ‘nature of the environment’

- QL overestimates → **‘optimistic’**. Hence, performs well in environments where **optimism is beneficial**!
- DQL underestimates → **‘pessimistic’**. Hence, performs well in environments where **pessimism is beneficial**!
- In the general case, the performance of QL and DQL is highly dependent on the ‘environment nature’.

QL, DQL and the ‘nature of the environment’

- QL overestimates → **‘optimistic’**. Hence, performs well in environments where **optimism is beneficial**!
- DQL underestimates → **‘pessimistic’**. Hence, performs well in environments where **pessimism is beneficial**!
- In the general case, the performance of QL and DQL is highly dependent on the ‘environment nature’.

What if we do not have **prior knowledge** about the environment nature?

Environment that involves both 'optimistic' and 'pessimistic' scenarios.

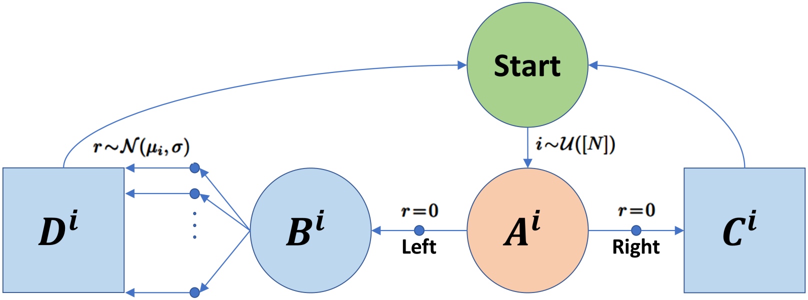
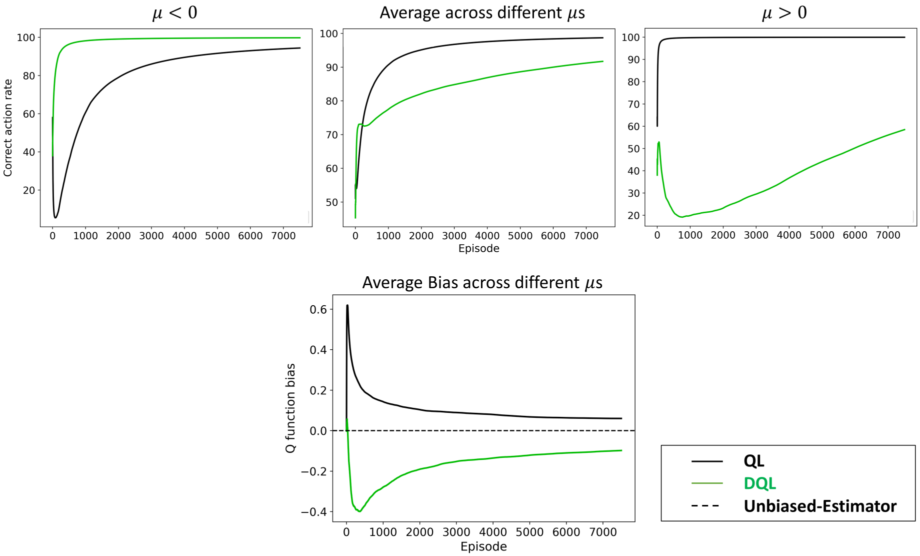


Figure: Meta Chain MDP

QL, DQL and the 'nature of the environment'



Statistical framework - Estimating the Maximum Expected Value

The *future reward* is a random variable, $R^\pi(s) = \sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a \sim \pi(s_t)$.

Denoting: $(X_1, \ldots, X_m) \triangleq (R^\pi(s_{t+1}, a_1), \ldots, R^\pi(s_{t+1}, a_m)),$
$(\mu_1, \ldots, \mu_m) \triangleq (Q^\pi(s_{t+1}, a_1), \ldots, Q^\pi(s_{t+1}.a_m)),$

The **next-state value** used by QL is determined by:

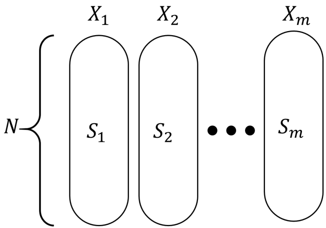
$$Q^*(s_{t+1}, a^*) = \mu^* = \max_a \mu_a.$$

Statistical framework - Estimating the Maximum Expected Value

Problem: estimate the *maximal expected value* of m independent random variables $\{X_1, \ldots, X_m\}$, with means $\{\mu_1, \ldots, \mu_m\}$:

$$\max_a \mathbb{E}[X_a] = \max_a \mu_a \triangleq \mu^* \quad ,$$

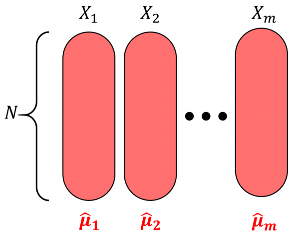
Based on N i.i.d. samples from the same distribution as $\{X\}_{i=1}^m$: $\{S\}_{i=1}^m$



Single Estimator and Double Estimator

Single Estimator (SE): Use the maximal empirical mean of the samples.

$$\hat{\mu}^*_{\text{SE}} \triangleq \max_a \hat{\mu}_a(S_a) = \max_a \frac{1}{N}\sum_{j=1}^{N} S_a(j)\,.$$



Single Estimator and Double Estimator

Proposition [Smith and Winkler 2006]

SE is proved to **overestimate** μ^*.

Single Estimator and Double Estimator

Proposition [Smith and Winkler 2006]

SE is proved to **overestimate** μ^*.

Proof:

- Let $a^* \in \arg\max_{a \in [m]} \mu_a$, and $\hat{a}^* = \arg\max_{a \in [m]} \hat{\mu}_a$
- SE is unbiased: $\mathbb{E}\left[\mu_{a^*}\right] = \hat{\mu}_{a^*}$

Single Estimator and Double Estimator

Proposition [Smith and Winkler 2006]

SE is proved to **overestimate** μ^*.

Proof:

- Let $a^* \in \arg\max_{a \in [m]} \mu_a$, and $\hat{a}^* = \arg\max_{a \in [m]} \hat{\mu}_a$
- SE is unbiased: $\mathbb{E}\left[\mu_{a^*}\right] = \hat{\mu}_{a^*}$
- Then: $\hat{\mu}_{a^*} \leq \hat{\mu}_{\hat{a}^*}$. And also $\mu_{a^*} - \hat{\mu}_{\hat{a}^*} \leq \mu_{a^*} - \hat{\mu}_{a^*}$

Single Estimator and Double Estimator

Proposition [Smith and Winkler 2006]

SE is proved to **overestimate** μ^*.

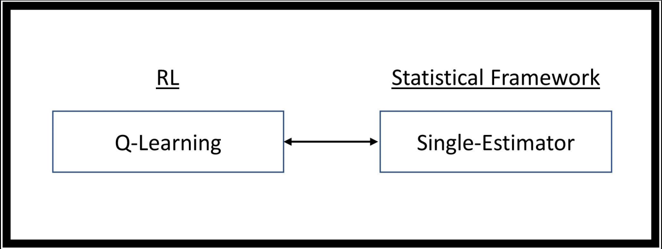
Proof:

- Let $a^* \in \arg\max_{a \in [m]} \mu_a$, and $\hat{a}^* = \arg\max_{a \in [m]} \hat{\mu}_a$
- SE is unbiased: $\mathbb{E}\left[\mu_{a^*}\right] = \hat{\mu}_{a^*}$
- Then: $\hat{\mu}_{a^*} \leq \hat{\mu}_{\hat{a}^*}$. And also $\mu_{a^*} - \hat{\mu}_{\hat{a}^*} \leq \mu_{a^*} - \hat{\mu}_{a^*}$
- Monotonicity of the expectation: $\mathbb{E}\left[\mu_{a^*} - \hat{\mu}_{\hat{a}^*}\right] \leq \mathbb{E}\left[\mu_{a^*} - \hat{\mu}_{a^*}\right] = 0$ ■

Single Estimator and Double Estimator

Proposition [Smith and Winkler 2006]

SE is proved to **over**



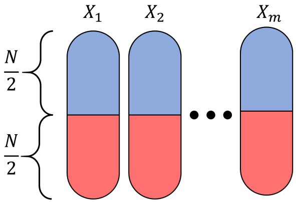
Proof:

- Let a^* ∈ arg max
- SE is unbiased:
- Then: $\hat{\mu}_{a^*} \leq \hat{\mu}_{\hat{a}^*}$
- Monotonicity of the expectation: $\mathbb{E}\left[\mu_{a^*} - \mu_{\hat{a}^*}\right] \leq \mathbb{E}\left[\mu_{a^*} - \mu_{a^*}\right] = 0$ ■

Single Estimator and Double Estimator

Double Estimator (DE): The samples of each random variable $a \in [m]$ are split into two disjoint, equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$

2-phase estimation process:

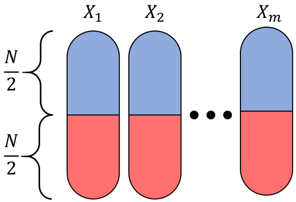


Single Estimator and Double Estimator

Double Estimator (DE): The samples of each random variable $a \in [m]$ are split into two disjoint, equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$

2-phase estimation process:

- The **index** $\hat{a}^*$ is estimated using $S^{(1)}$: $\hat{a}^* = \text{argmax}_a \, \hat{\mu}_a^{(1)}$.

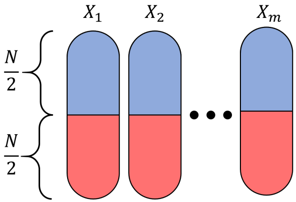


Single Estimator and Double Estimator

Double Estimator (DE): The samples of each random variable $a \in [m]$ are split into two disjoint, equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$

2-phase estimation process:

- The **index** $\hat{a}^*$ is estimated using $S^{(1)}$: $\hat{a}^* = \text{argmax}_a \hat{\mu}_a^{(1)}$.
- The **mean** of $X_{\hat{a}^*}$ is estimated using $S_{\hat{a}^*}^{(2)}$: $\hat{\mu}_{\text{DE}}^* = \hat{\mu}^{(2)}{}_{\hat{a}^*}$.



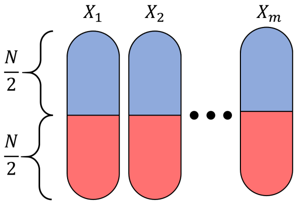
Single Estimator and Double Estimator

Double Estimator (DE): The samples of each random variable $a \in [m]$ are split into two disjoint, equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$

2-phase estimation process:

- The **index** $\hat{a}^*$ is estimated using $S^{(1)}$: $\hat{a}^* = \text{argmax}_a \, \hat{\mu}_a^{(1)}$.
- The **mean** of $X_{\hat{a}^*}$ is estimated using $S_{\hat{a}^*}^{(2)}$: $\hat{\mu}_{\text{DE}}^* = \hat{\mu}^{(2)}{}_{\hat{a}^*}$.

DE is proved to **underestimate** μ^*.[3]



[3]Van Hasselt 2010.

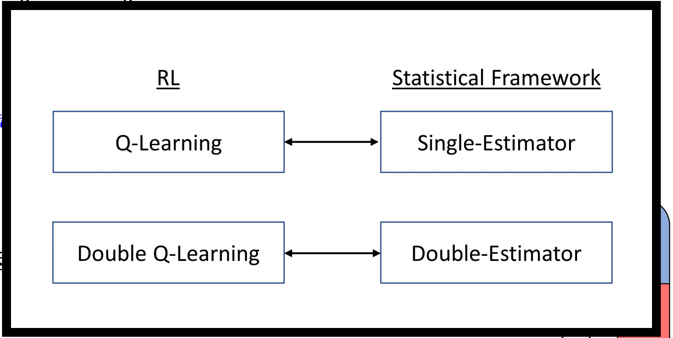
Single Estimator and Double Estimator

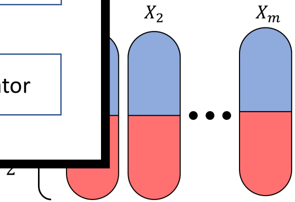
Double Estimator (DE): The samples of each random variable $a \in [m]$ are split into two disjoint, equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$

2-phase estimation

- The **index** $\hat{a}^*$ is
- The **mean** of $X_{\hat{a}}$

DE is proved to **unde**

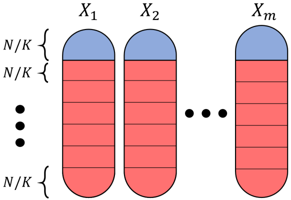




[3]Van Hasselt 2010.

Ensemble Estimator (EE)

Ensemble Estimator (EE): The samples of each random variable $a \in [m]$ are split into K disjoint, equal-sized subsets $\left\{S_a^{(i)}\right\}_{i=1}^{K}$.
2-phase estimation process:

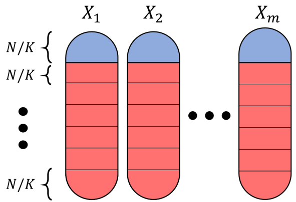


Ensemble Estimator (EE)

Ensemble Estimator (EE): The samples of each random variable $a \in [m]$ are split into K disjoint, equal-sized subsets $\left\{S_a^{(i)}\right\}_{i=1}^{K}$.

2-phase estimation process:

- A single arbitrary set $\tilde{k} \in [K]$ is used to estimate the **index**: $\hat{a}^* = \text{argmax}_a \hat{\mu}_a^{(\tilde{k})}$.



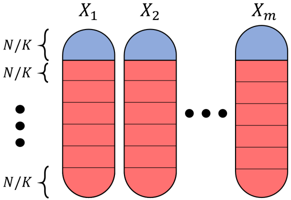
Ensemble Estimator (EE)

Ensemble Estimator (EE): The samples of each random variable $a \in [m]$ are split into K disjoint, equal-sized subsets $\left\{S_a^{(i)}\right\}_{i=1}^{K}$.

2-phase estimation process:

- A single arbitrary set $\tilde{k} \in [K]$ is used to estimate the **index**: $\hat{a}^* = \operatorname{argmax}_a \hat{\mu}_a^{(\tilde{k})}$.
- the **mean** is estimated using the rest of the ensemble:

$$\hat{\mu}^*_{\mathrm{EE}} \triangleq \frac{1}{K-1} \sum_{j \in [K] \setminus \tilde{k}} \hat{\mu}_{\hat{a}^*}^{(j)}.$$

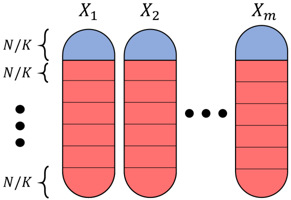


Ensemble Estimator (EE)

Ensemble Estimator (EE): The samples of each random variable $a \in [m]$ are split into K disjoint, equal-sized subsets $\left\{S_a^{(i)}\right\}_{i=1}^{K}$.
2-phase estimation process:

- A single arbitrary set $\tilde{k} \in [K]$ is used to estimate the **index**: $\hat{a}^* = \text{argmax}_a \hat{\mu}_a^{(\tilde{k})}$.
- the **mean** is estimated using the rest of the ensemble:

$$\hat{\mu}_{\text{EE}}^* \triangleq \frac{1}{K-1} \sum_{j \in [K] \setminus \tilde{k}} \hat{\mu}_{\hat{a}^*}^{(j)}.$$



Note that $K = 2 \iff$ DE=EE

Ensemble Estimator (EE)

Ensemble Estimator ... split into K disjoint,

equal-sized subsets

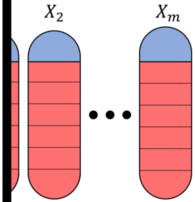
2-phase estimation

- A single arbitra... $_a\,\hat{\mu}_a^{(\tilde{k})}$.
- the **mean** is esti...

$\hat{\mu}^*_{\text{EE}} \triangleq \frac{1}{K-1}\sum_{j\in[K}$

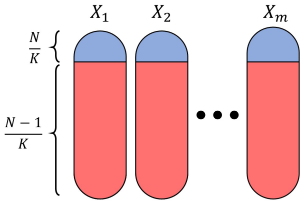
Note that $K = 2$

RL		Statistical Framework
Q-Learning	⟷	Single-Estimator
Double Q-Learning	⟷	Double-Estimator
Ensemble Bootstrapped Q-Learning	⟷	Ensemble-Estimator



Weighted Double Estimator

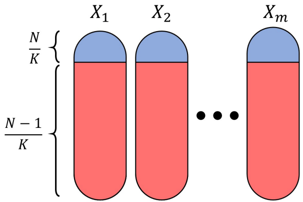
Weighted Double Estimator (W-DE): similar to DE, **BUT**, the samples of each random variable $a \in [m]$ are split into two disjoint, **not**-equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$, where $|S_a^{(1)}| < |S_a^{(2)}|$.
2-phase estimation process:



Weighted Double Estimator

Weighted Double Estimator (W-DE): similar to DE, **BUT**, the samples of each random variable $a \in [m]$ are split into two disjoint, **not**-equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$, where $|S_a^{(1)}| < |S_a^{(2)}|$.
2-phase estimation process:

- The **index** $\hat{a}^*$ is estimated using $S^{(1)}$: $\hat{a}^* = \text{argmax}_a \hat{\mu}_a^{(1)}$.

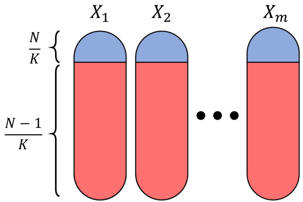


Weighted Double Estimator

Weighted Double Estimator (W-DE): similar to DE, **BUT**, the samples of each random variable $a \in [m]$ are split into two disjoint, **not**-equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$, where $|S_a^{(1)}| < |S_a^{(2)}|$.

2-phase estimation process:

- The **index** $\hat{a}^*$ is estimated using $S^{(1)}$: $\hat{a}^* = \text{argmax}_a \, \hat{\mu}_a^{(1)}$.
- The **mean** of $X_{\hat{a}^*}$ is estimated using $S_{\hat{a}^*}^{(2)}$: $\hat{\mu}_{\text{W-DE}}^* = \hat{\mu}^{(2)}{}_{\hat{a}^*}$.



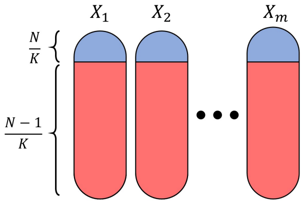
Weighted Double Estimator

Weighted Double Estimator (W-DE): similar to DE, **BUT**, the samples of each random variable $a \in [m]$ are split into two disjoint, **not**-equal-sized subsets $S_a^{(1)}$ and $S_a^{(2)}$, where $|S_a^{(1)}| < |S_a^{(2)}|$.

2-phase estimation process:

- The **index** $\hat{a}^*$ is estimated using $S^{(1)}$: $\hat{a}^* = \text{argmax}_a \, \hat{\mu}_a^{(1)}$.
- The **mean** of $X_{\hat{a}^*}$ is estimated using $S_{\hat{a}^*}^{(2)}$: $\hat{\mu}^*_{\text{W-DE}} = \hat{\mu}^{(2)}{}_{\hat{a}^*}$.

W-DE **underestimates** μ^* same as DE.



Weighted Double Estimator $\iff$ Ensemble Estimator

Proposition - Proxy Identity

$\hat{\mu}^*_{\text{EE}} = \hat{\mu}^*_{\text{W-DE}}$

W-DE offers much easier formulation for analysis! Yet less appealing for RL.

Weighted Double Estimator $\Longleftrightarrow$ Ensemble Estimator

Proposition - SNR and suboptimal ratio

Let $X = (X_1, X_2) \sim \mathcal{N}\left((\mu_1, \mu_2)^T, \sigma^2 I_2\right)$ be a Gaussian random vector such that $\mu_1 \geq \mu_2$ and let $\Delta = \mu_1 - \mu_2$. Also, define the signal to noise ratio as $\text{SNR} = \frac{\Delta}{\sigma/\sqrt{N}}$ and let $\hat{\mu}^*_{\text{W-DE}}$ be a W-DE that uses N_1 samples for index estimation. Then, for any fixed even sample-size $N > 10$ and any N_1^* that minimizes $\text{MSE}(\hat{\mu}^*_{\text{W-DE}})$, it holds that:

(1) As $\text{SNR} \to \infty$, $N_1^* \to 1$

(2) As $\text{SNR} \to 0$, $N_1^* \to 1$

(3) For any σ and Δ, it holds that $N_1^* < N/2$.

Weighted Double Estimator $\Longleftrightarrow$ Ensemble Estimator

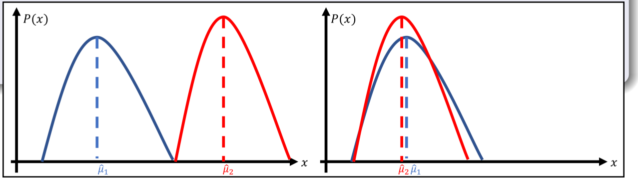
Proposition - SNR and suboptimal ratio

Let $X = (X_1, X_2) \sim \mathcal{N}\left((\mu_1, \mu_2)^T, \sigma^2 I_2\right)$ be a Gaussian random vector such that $\mu_1 \geq \mu_2$ and let $\Delta = \mu_1 - \mu_2$. Also, define the signal to noise ratio as $\text{SNR} = \frac{\Delta}{\sigma/\sqrt{N}}$ and let $\hat{\mu}^*_{\text{W-DE}}$ be a W-DE that uses N_1 samples for index estimation. Then, for any fixed even sample-size $N > 10$ and any N_1^* that minimizes $\text{MSE}(\hat{\mu}^*_{\text{W-DE}})$, it holds that:

(1) As $\text{SNR} \to \infty$, $N_1^* \to 1$

(2) As $\text{SNR} \to 0$, $N_1^* \to 1$

(3) For any σ and Δ, it holds that $N_1^* < N/2$.



Weighted Double Estimator

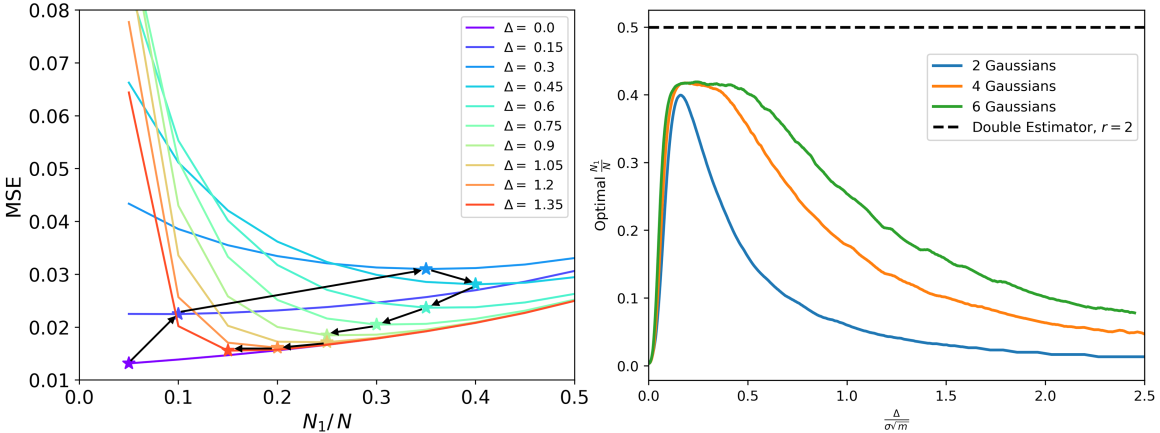


Figure: Left: MSE as function of N_1. Right: Optimal split-ratio as function of SNR

Ensemble Estimator (EE)

- We show that when facing **LOW** SNR (variables are hard to distinguish) it is MSE-beneficial to use large ensembles.

Ensemble Estimator (EE)

- We show that when facing **LOW** SNR (variables are hard to distinguish) it is MSE-beneficial to use large ensembles.
- RL is considered as **noisy** environment, hence we expect large EE-like algorithms will reduce the MSE of the Q-function estimation.

Algorithm 2: Ensemble Bootstrapped Q-Learning (EBQL)

Initialize: K Q-tables: $\left\{Q^i\right\}_{i=1}^{K}$

for $t = 1, \ldots, T$ **do**

Choose action $a_t = \operatorname{argmax}_a \left[\sum_{i=1}^{K} Q^i(s_t, a)\right]$

$a_t = \text{explore}(a_t)$; // e.g. ϵ-greedy

$s_{t+1}, r_t \leftarrow \text{env.step}(s_t, a_t)$

Sample an ensemble member to update: $k_t \sim \mathcal{U}([K])$

Define $\hat{a}^* = \operatorname{argmax}_a Q^{k_t}(s_{t+1}, a)$

$Q^{k_t}(s_t, a_t) \leftarrow (1 - \alpha_t) Q^{k_t}(s_t, a_t) + \alpha_t \left(r_t + \gamma Q^{EN \setminus k_t}(s_{t+1}, \hat{a}^*)\right)$

Result: $\left\{Q^i\right\}_{i=1}^{K}$

Where $Q^{EN\setminus k_t}(s_{t+1}, \hat{a}^*) = \frac{1}{K-1} \sum_{j \in [K] \setminus k_t} Q^j(s_{t+1}, \hat{a}^*)$.

Short Summary

RL		Statistical Framework
Q-Learning	⟷	Single-Estimator
Double Q-Learning	⟷	Double-Estimator
Ensemble Bootstrapped Q-Learning	⟷	Ensemble-Estimator

Results: Meta Chain MDP

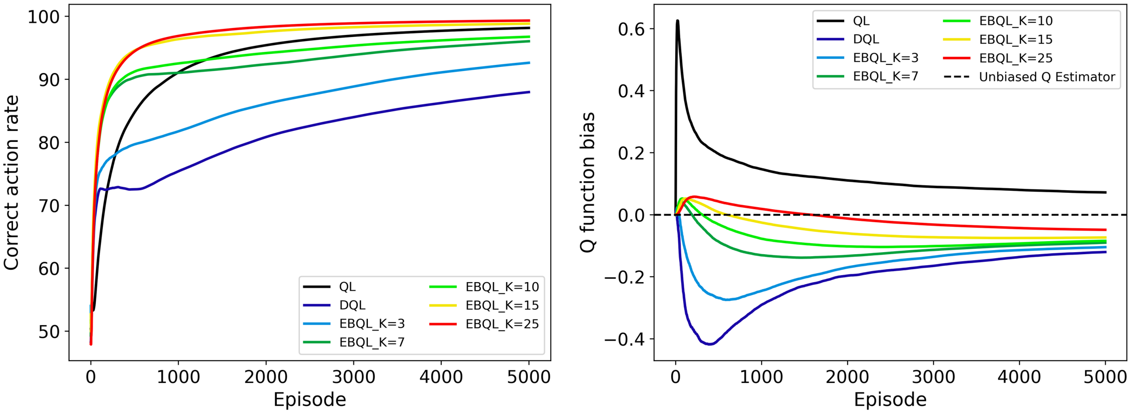


Figure: EBQL Vs. QL and DQL

Other Ensemble-based works

- Avg. DQN - Anschel, Baram, and Shimkin 2017 - Use the *entire* ensemble to approximate the next-state Q-function (extension of QL):

Other Ensemble-based works

- Avg. DQN - Anschel, Baram, and Shimkin 2017 - Use the *entire* ensemble to approximate the next-state Q-function (extension of QL):

$$\forall i \in [K], \quad TD^i_{Avg}(s_t, a_t) = r_t + \gamma \max_a \left[\frac{1}{K} \sum_k Q^k(s_{t+1}, a) \right] - Q^i(s_t, a_t)$$

Other Ensemble-based works

- Avg. DQN - Anschel, Baram, and Shimkin 2017 - Use the *entire* ensemble to approximate the next-state Q-function (extension of QL):

$$\forall i \in [K], \quad TD^i_{Avg}(s_t, a_t) = r_t + \gamma \max_a \left[\frac{1}{K} \sum_k Q^k(s_{t+1}, a) \right] - Q^i(s_t, a_t)$$

  - Reduces the variance of the target approximation error.

Other Ensemble-based works

- Avg. DQN - Anschel, Baram, and Shimkin 2017 - Use the *entire* ensemble to approximate the next-state Q-function (extension of QL):

$$\forall i \in [K] , \quad TD^i_{Avg}(s_t, a_t) = r_t + \gamma \max_a \left[\frac{1}{K} \sum_k Q^k(s_{t+1}, a) \right] - Q^i(s_t, a_t)$$

  - Reduces the variance of the target approximation error.
  - Still Overestimates.

Other Ensemble-based works

- Avg. DQN - Anschel, Baram, and Shimkin 2017 - Use the *entire* ensemble to approximate the next-state Q-function (extension of QL):

$$\forall i \in [K] , \quad TD^{i}_{Avg}(s_t, a_t) = r_t + \gamma \max_{a} \left[\frac{1}{K} \sum_{k} Q^{k}(s_{t+1}, a) \right] - Q^{i}(s_t, a_t)$$

  - Reduces the variance of the target approximation error.
  - Still Overestimates.
- MaxMin Q-Learning, Lan et al. 2020 - Construct a **'pessimist'** target: $Q_{min}(s, a) = \min_{K \in [K]} Q^{k}(s, a), \ \forall a.$

Other Ensemble-based works

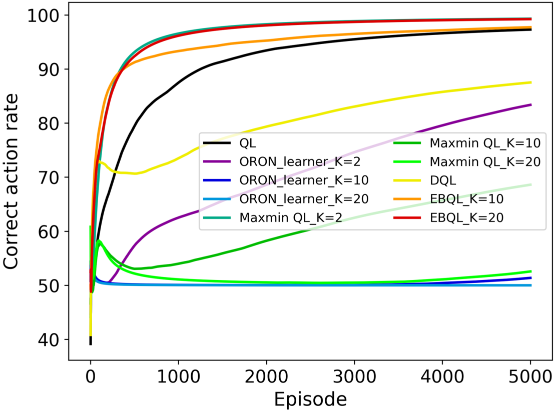
- Avg. DQN - Anschel, Baram, and Shimkin 2017 - Use the *entire* ensemble to approximate the next-state Q-function (extension of QL):

$$\forall i \in [K] , \quad TD^i_{Avg}(s_t, a_t) = r_t + \gamma \max_a \left[\frac{1}{K} \sum_k Q^k(s_{t+1}, a) \right] - Q^i(s_t, a_t)$$

  - Reduces the variance of the target approximation error.
  - Still Overestimates.
- MaxMin Q-Learning, Lan et al. 2020 - Construct a '**pessimist**' target: $Q_{min}(s, a) = \min_{K \in [K]} Q^k(s, a), \ \forall a.$

$$\forall i \in [K] , \quad TD^i_{maxMin}(s_t, a_t) = r_t + \gamma \max_a Q_{min}(s_{t+1}, a) - Q^i(s_t, a_t)$$

Results: Meta Chain MDP



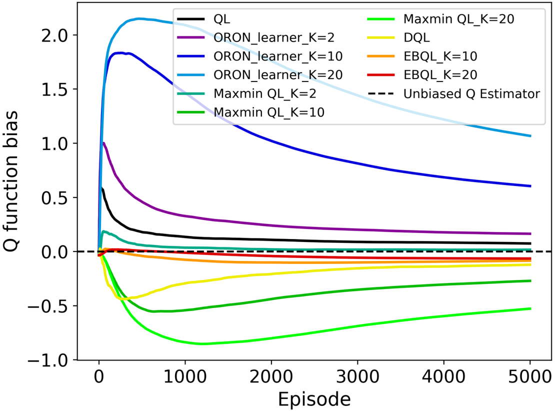


Figure: EBQL Vs. QL, DQL, Avg. QL and MaxMin-QL

Arcade Learning Environment (ALE) - Atari

Arcade Learning Environment (ALE) - Atari

Figure: Atari Console: Input image-size: 160X210 Pixels, 18 discrete actions defined by the joystick controller.

Results: Atari

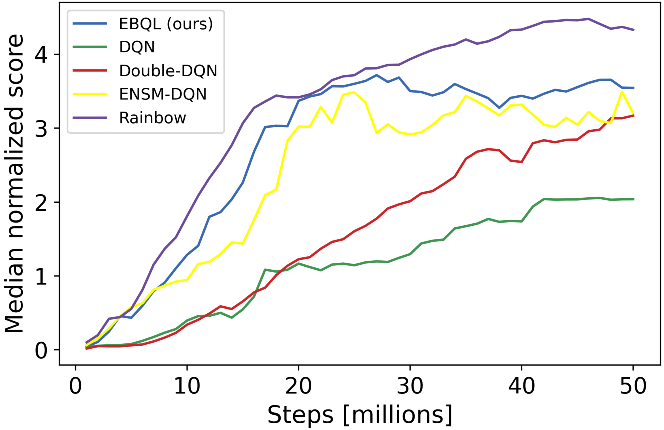


Figure: Comparison of the DQN, DDQN, Rainbow[4] and EBQL agents on 11 random ATARI environments.

[4] Hessel et al. 2018.

Summary and future work

- QL - Overestimates. Might be harmful sometimes.

Summary and future work

- QL - Overestimates. Might be harmful sometimes.
- DQL - solves overestimation. Results with underestimating. ***Widely used across SOTA algorithms***.

Summary and future work

- QL - Overestimates. Might be harmful sometimes.
- DQL - solves overestimation. Results with underestimating. ***Widely used across SOTA algorithms***.
- **We can do (MSE) better! - EBQL** - reduces the MSE of next-state Q-function using better ‘budget’ split between *index* and *mean* estimations.

Summary and future work

- QL - Overestimates. Might be harmful sometimes.
- DQL - solves overestimation. Results with underestimating. ***Widely used across SOTA algorithms***.
- **We can do (MSE) better! - EBQL** - reduces the MSE of next-state Q-function using better 'budget' split between *index* and *mean* estimations.

Future work:

Summary and future work

- QL - Overestimates. Might be harmful sometimes.
- DQL - solves overestimation. Results with underestimating. ***Widely used across SOTA algorithms***.
- **We can do (MSE) better! - EBQL** - reduces the MSE of next-state Q-function using better ‘budget’ split between *index* and *mean* estimations.

Future work:

- Improve DQL-based SOTA algorithms using EBQL.

Summary and future work

- QL - Overestimates. Might be harmful sometimes.
- DQL - solves overestimation. Results with underestimating. ***Widely used across SOTA algorithms***.
- **We can do (MSE) better! - EBQL** - reduces the MSE of next-state Q-function using better ‘budget’ split between *index* and *mean* estimations.

Future work:

- Improve DQL-based SOTA algorithms using EBQL.
- Convergence, rates, optimal split ratio.

Summary and future work

- QL - Overestimates. Might be harmful sometimes.
- DQL - solves overestimation. Results with underestimating. ***Widely used across SOTA algorithms***.
- **We can do (MSE) better! - EBQL** - reduces the MSE of next-state Q-function using better 'budget' split between *index* and *mean* estimations.

Future work:

- Improve DQL-based SOTA algorithms using EBQL.
- Convergence, rates, optimal split ratio.
- Dynamic split-ratio using values+variance estimations (we already have an ensemble..)

Thank You.

References

Anschel, Oron, Nir Baram, and Nahum Shimkin (2017). “Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning”. In: *International Conference on Machine Learning*. PMLR, pp. 176–185.

Hessel, Matteo et al. (2018). “Rainbow: Combining improvements in deep reinforcement learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32.

Lan, Qingfeng et al. (2020). “Maxmin q-learning: Controlling the estimation bias of q-learning”. In: *arXiv preprint arXiv:2002.06487*.

Smith, James E and Robert L Winkler (2006). “The optimizer’s curse: Skepticism and postdecision surprise in decision analysis”. In: *Management Science* 52.3, pp. 311–322.

Thrun, Sebastian and Anton Schwartz (1993). “Issues in using function approximation for reinforcement learning”. In: *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*.

Van Hasselt, Hado (2010). “Double Q-learning”. In: *Advances in neural information processing systems* 23, pp. 2613–2621.

Van Hasselt, Hado et al. (2018). “Deep reinforcement learning and the deadly triad”. In: *arXiv preprint arXiv:1812.02648*.

Watkins, Christopher JCH and Peter Dayan (1992). “Q-learning”. In: *Machine learning* 8.3-4, pp. 279–292.