Deep Q-Networks (DQN)

Insoon Yang

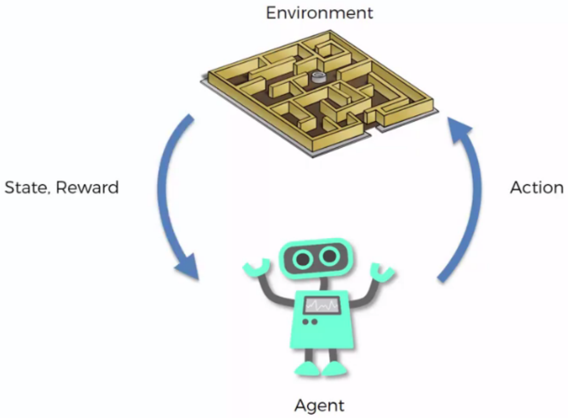
Department of Electrical and Computer Engineering
Seoul National University

Control + Optimization Research Lab

Review: MDP

- It is a framework for an agent's sequential decision-making by interacting with an uncertain environment

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$



Review: Q-function

Q) What is Q-function?

Review: Q-function

Q) What is Q-function?

- $Q(s, a)$ is the maximal expected cumulative reward starting from state s, taking action a.

Review: Q-function

Q) What is Q-function?

- $Q(s, a)$ is the maximal expected cumulative reward starting from state s, taking action a.

Q) Why do we care about Q-function?

Review: Q-function

Q) What is Q-function?

- $Q(s, a)$ is the maximal expected cumulative reward starting from state s, taking action a.

Q) Why do we care about Q-function?

- We can use it to obtain an optimal policy.

Review: Q-function

Q) What is Q-function?

- $Q(s, a)$ is the maximal expected cumulative reward starting from state s, taking action a.

Q) Why do we care about Q-function?

- We can use it to obtain an optimal policy.

Q) How?

Review: Q-function

Q) What is Q-function?

- $Q(s, a)$ is the maximal expected cumulative reward starting from state s, taking action a.

Q) Why do we care about Q-function?

- We can use it to obtain an optimal policy.

Q) How?

- Choose an $a^* \in \max_a Q(s, a)$. Then, select

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = a^* \\ 0 & \text{otherwise} \end{cases}$$

Review: Computing Q-function

Q) Ok, then how can we compute Q-function?

Review: Computing Q-function

Q) Ok, then how can we compute Q-function?

- Solve the Bellman equation:

$$Q(s,a) = \underbrace{r(s,a)}_{\text{immediate reward}} + \gamma \underbrace{\mathbb{E}_{s'}\left[\max_{a'} Q(s',a')\right]}_{\text{optimal value of next state}}$$

$$= r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a'} Q(s',a')$$

Review: Computing Q-function

Q) Ok, then how can we compute Q-function?

- Solve the Bellman equation:

$$
\begin{aligned}
Q(s,a) &= \underbrace{r(s,a)}_{\text{immediate reward}} + \gamma \underbrace{\mathbb{E}_{s'}\left[\max_{a'} Q(s',a')\right]}_{\text{optimal value of next state}} \\
&= r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a'} Q(s',a')
\end{aligned}
$$

It's a fixed point problem:

$$Q = FQ$$

Review: Value Iteration using Q-function

Q) How can we solve the Bellman equation?

Review: Value Iteration using Q-function

Q) How can we solve the Bellman equation?

- The simplest way is to use value iteration

Review: Value Iteration using Q-function

Q) How can we solve the Bellman equation?

- The simplest way is to use value iteration

Initialize Q_0;

1. Set $Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'}\left[\max_{a'} Q_k(s',a')\right]$;
2. Set $k \leftarrow k+1$; Repeat until convergence;

Review: Value Iteration using Q-function

Q) How can we solve the Bellman equation?

- The simplest way is to use value iteration

Initialize Q_0;

1. Set $Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'}\left[\max_{a'} Q_k(s',a')\right]$;
2. Set $k \leftarrow k+1$; Repeat until convergence;

But this approach requires MDP model...

Review: Q-Learning

Q) What can we do if we do not know MDP model?

Review: Q-Learning

Q) What can we do if we do not know MDP model?

- **Use sample data (experience)** (s, a, s', r)**: Q-learning**

Review: Q-Learning

Q) What can we do if we do not know MDP model?

- **Use sample data (experience)** (s, a, s', r)**: Q-learning**

Initialize Q;

1. Take some action and observe (s, a, s', r);
2. Set $Q(s,a) \leftarrow (1-\alpha) \underbrace{Q(s,a)}_{\text{old estimate}} + \alpha \underbrace{\left[r + \gamma \max_{a'} Q(s', a')\right]}_{\text{new estimate}}$;
3. Repeat until convergence;

Review: Q-Learning

Q) What can we do if we do not know MDP model?

- **Use sample data (experience)** (s, a, s', r)**: Q-learning**

Initialize Q;

1. Take some action and observe (s, a, s', r);
2. Set $Q(s,a) \leftarrow (1-\alpha) \underbrace{Q(s,a)}_{\text{old estimate}} + \alpha \underbrace{\left[r + \gamma \max_{a'} Q(s', a')\right]}_{\text{new estimate}}$;
3. Repeat until convergence;

Note:

- (s, a, s') gives information about transition $p(s'|s, a)$
- (s, a, r) gives information about reward $r(s, a)$

Review: Approximate Q-Learning

Q) How can we solve large-scale problems?

Review: Approximate Q-Learning

Q) How can we solve large-scale problems?

- **Parameterize Q-function:** $Q_\phi(s, a)$**, where** ϕ **is a parameter vector**

Review: Approximate Q-Learning

Q) How can we solve large-scale problems?

- **Parameterize Q-function:** $Q_\phi(s,a)$**, where** ϕ **is a parameter vector**

Initialize ϕ;

1. Collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy;
2. For $i = 1 : N$
   - Set $\underbrace{y_i}_{\text{target}} \leftarrow \underbrace{r_i + \gamma \max_a Q_\phi(s'_i, a)}_{\text{new estimate}}$;
3. Set $\phi \leftarrow \arg\min_\phi \underbrace{\frac{1}{2}\sum_i \|Q_\phi(s_i, a_i) - y_i\|^2}_{\text{loss function}}$;
4. Repeat until convergence;

Review: Approximate Q-Learning (online version)

Initialize ϕ;

1. Take some action and observe (s_i, a_i, s_i', r_i);
2. Set Set $\underbrace{y_i}_{\text{target}} \leftarrow \underbrace{r_i + \gamma \max_a Q_\phi(s_i', a)}_{\text{new estimate}}$;
3. Set $\phi \leftarrow \phi - \underbrace{\alpha}_{\text{stepsize}} \underbrace{\frac{dQ_\phi}{d\phi}(Q_\phi(s_i, a_i) - y_i)}_{\text{stochastic gradient}}$;
4. Repeat until convergence;

Advantages and Disadvantages

Q) What are the advantages of Q-learning?

- Simple
- Off-policy: Can use any policy to generate samples
- Some useful theory

Advantages and Disadvantages

Q) What are the advantages of Q-learning?

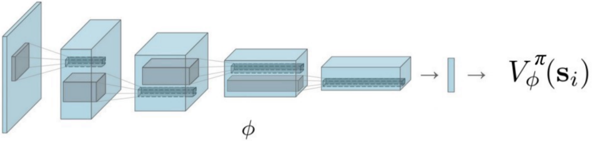
- Simple
- Off-policy: Can use any policy to generate samples
- Some useful theory

Q) What are disadvantages?

- Unclear how to parameterize
- Correlation between samples
- Overestimation
- Exploration

DQN = Q-Learning + Deep Learning

- Q-Learning: optimal behaviors
- Deep NN: unstructured environments, complex sensory inputs



Success I: Game

Success II: Robotics

- 580k real-world grasp attempts to train a deep neural network Q-function with over 1.2M parameters
- 96% grasp success on unseen objects

Can we just use NN to approximate Q-function?

European Conference on Machine Learning

ECML 2005: Machine Learning: ECML 2005 pp 317-328 | Cite as

Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method

Authors | Authors and affiliations

Martin Riedmiller

Can we just use NN to approximate Q-function?

European Conference on Machine Learning
ECML 2005: Machine Learning: ECML 2005 pp 317-328 | Cite as

Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method

Authors | Authors and affiliations

Martin Riedmiller

No. Some issues:

- Samples (sequential states) are correlated
  $\Longrightarrow$ Producing bias
- Target value keeps changing
  $\Longrightarrow$ Inconsistent and unstable training

Two Innovations in DQN

Very simple ideas:

Two Innovations in DQN

Very simple ideas:

- **Experience replay (Replay buffer)**

Two Innovations in DQN

Very simple ideas:

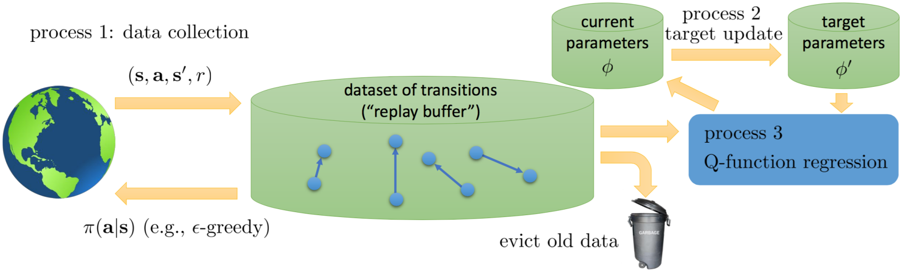
- **Experience replay (Replay buffer)** sample correlation 타개
- Separate target network
  $Q_{\phi-}$: **target (updated slowly)**
  Q_{ϕ}: current value

DQN Algorithm

Initialize ϕ and ϕ^-;

1. Save target network parameters: $\phi^- \leftarrow \phi$;
2. Repeat N times
    1. Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy, add them to **Buffer**;
    2. Repeat K times
        1. **Sample a mini-batch** $\{(s_j, a_j, s_j', r_j)\}$ from **Buffer**;
           randomly make minibatch (correlation 붕괴)
        2. Compute the **target** $y_j^- := r_j + \gamma \max_a Q_{\phi^-}(s_j', a)$ for all j;
        3. Update $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(Q_\phi(s_j, a_j) - y_j^-)$;
           Not Qphi But Qphi - as fixed
3. Repeat until converges;

DQN Algorithm



[S. Levine, CS285]

Comparison:

- Online Q-learning: Processes 1, 2, 3 all run at the same speed
- DQN: Processes 1, 3 run at the same speed, Process 2 is slow

Dissecting DQN I: Data collection and sampling

Experience Replay:

- Transition data (s, a, s', r) are added into **Buffer**.
- They are randomly sampled to update Q-function.

Dissecting DQN I: Data collection and sampling

Experience Replay:

- Transition data (s, a, s', r) are added into **Buffer**.
- They are randomly sampled to update Q-function.

Q) What's the benefit of experience replay?

Dissecting DQN I: Data collection and sampling

Experience Replay:

- Transition data (s, a, s', r) are added into **Buffer**.
- They are randomly sampled to update Q-function.

Q) What's the benefit of experience replay?

- Break the correlation between samples (sequential states)
  $\Longrightarrow$ Reduce bias

Dissecting DQN II: Target update

Target parameters ϕ^- are synchronized as ϕ every N steps

Dissecting DQN II: Target update

Target parameters ϕ^{-} are synchronized as ϕ every N steps

Q) What's the benefit of using separate target network?

Dissecting DQN II: Target update

Target parameters ϕ^{-} are synchronized as ϕ every N steps

Q) What's the benefit of using separate target network?

- Can update ϕ by solving a regression problem with a fixed target $\implies$ Consistent and stable training

Dissecting DQN II: Target update

Target parameters ϕ^- are synchronized as ϕ every N steps

Q) What's the benefit of using separate target network?

- Can update ϕ by solving a regression problem with a fixed target $\implies$ Consistent and stable training

Alternative option: Polyak averaging

- Update every step:

$$\phi^- \leftarrow (1-\tau)\phi^- + \tau\phi$$

  for very small τ

Dissecting DQN III: Q-function regression

Given a mini-batch $\{(s_j, a_j, s'_j, r_j)\}$ sampled from **Buffer**, solve the **regression problem**:

$$\min_{\phi} \frac{1}{2} \sum_{j} \| Q_{\phi}(s_j, a_j) - \underbrace{y_j^-}_{\text{target Q}} \|^2$$

Dissecting DQN III: Q-function regression

Given a mini-batch $\{(s_j, a_j, s'_j, r_j)\}$ sampled from **Buffer**,
solve the **regression problem**:

$$\min_{\phi} \frac{1}{2} \sum_{j} \| Q_{\phi}(s_j, a_j) - \underbrace{y_j^{-}}_{\text{target Q}} \|^2$$

Q) How?

Dissecting DQN III: Q-function regression

Given a mini-batch $\{(s_j, a_j, s'_j, r_j)\}$ sampled from **Buffer**, solve the **regression problem**:

$$\min_{\phi} \frac{1}{2} \sum_j \|Q_\phi(s_j, a_j) - \underbrace{y_j^-}_{\text{target Q}} \|^2$$

Q) How?

- Stochastic gradient descent (SGD)

$$\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi} (Q_\phi(s_j, a_j) - y_j^-)$$

Dissecting DQN III: Q-function regression

Given a mini-batch $\{(s_j, a_j, s'_j, r_j)\}$ sampled from **Buffer**, solve the **regression problem**:

$$\min_\phi \frac{1}{2} \sum_j \|Q_\phi(s_j, a_j) - \underbrace{y_j^-}_{\text{target Q}} \|^2$$

Q) How?

- Stochastic gradient descent (SGD)

$$\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi} (Q_\phi(s_j, a_j) - y_j^-)$$

Q) What's the benefit of Q-function regression?

Dissecting DQN III: Q-function regression

Given a mini-batch $\{(s_j, a_j, s'_j, r_j)\}$ sampled from **Buffer**, solve the **regression problem**:

$$\min_{\phi} \frac{1}{2} \sum_j \|Q_\phi(s_j, a_j) - \underbrace{y_j^-}_{\text{target Q}} \|^2$$

Q) How?

- Stochastic gradient descent (SGD)

$$\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(Q_\phi(s_j, a_j) - y_j^-)$$

Q) What's the benefit of Q-function regression?

- Can use the good method for deep learning (e.g., backprop)

Advantages and Disadvantages

Advantages:

- Simple
- Sample efficiency (off-policy + experience replay)

Advantages and Disadvantages

Advantages:

- Simple
- Sample efficiency (off-policy + experience replay)

Disadvantages:

- Difficult to handle continuous spaces
- Exploration
- Overestimation

Practical Tips

- Which policy to use to generate samples?

Practical Tips

- Which policy to use to generate samples?
  ϵ-**Greedy**:
  - Choose $a_t \in \arg\max_a Q_\phi(s_t, a)$ with probability $(1 - \epsilon)$
  - Choose a random action with probability ϵ

Practical Tips

- Which policy to use to generate samples?
  ϵ-**Greedy**:
  - Choose $a_t \in \arg\max_a Q_\phi(s_t, a)$ with probability $(1 - \epsilon)$
  - Choose a random action with probability ϵ
  - Start with high ϵ and gradually reduce. *Q) Why?*

Practical Tips

- Which policy to use to generate samples?
  ϵ-**Greedy**:
  - Choose $a_t \in \arg\max_a Q_\phi(s_t, a)$ with probability $(1 - \epsilon)$
  - Choose a random action with probability ϵ
  - Start with high ϵ and gradually reduce. *Q) Why?*
- Use large replay buffer: to improve stability

Practical Tips

- Which policy to use to generate samples?
  ϵ-**Greedy**:
  - Choose $a_t \in \arg\max_a Q_\phi(s_t, a)$ with probability $(1 - \epsilon)$
  - Choose a random action with probability ϵ
  - Start with high ϵ and gradually reduce. *Q) Why?*
- Use large replay buffer: to improve stability
- Test on easy tasks first

Practical Tips

- Which policy to use to generate samples?
  ϵ-**Greedy**:
  - Choose $a_t \in \arg\max_a Q_\phi(s_t, a)$ with probability $(1-\epsilon)$
  - Choose a random action with probability ϵ
  - Start with high ϵ and gradually reduce. *Q) Why?*
- Use large replay buffer: to improve stability
- Test on easy tasks first
- Be patient

Practical Tips

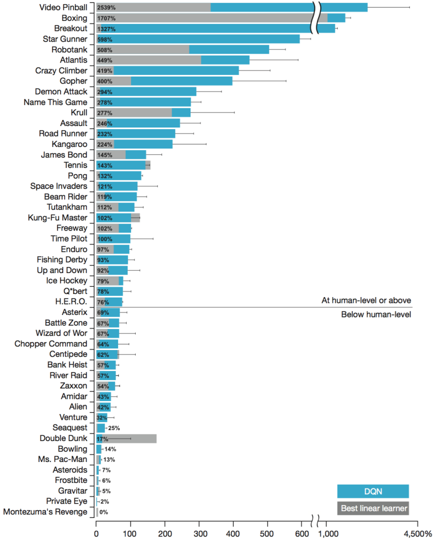
- Which policy to use to generate samples?
  ϵ-**Greedy**:
  - Choose $a_t \in \arg\max_a Q_\phi(s_t, a)$ with probability $(1 - \epsilon)$
  - Choose a random action with probability ϵ
  - Start with high ϵ and gradually reduce. *Q) Why?*
- Use large replay buffer: to improve stability
- Test on easy tasks first
- Be patient
- Bellman error gradients can be big:
  Clip gradients or use Huber loss

Results I: Mnih et al., 2013

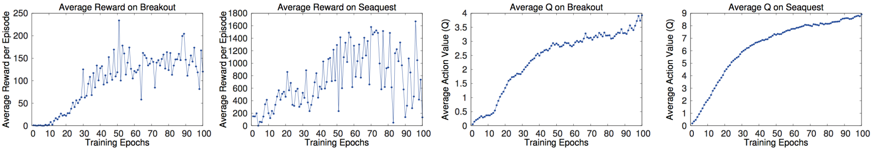
	B. Rider	**Breakout**	**Enduro**	**Pong**	**Q*bert**	**Seaquest**	**S. Invaders**
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Results II: Mnih et al., 2015

- Human vs DQN vs best RL methods before DQN

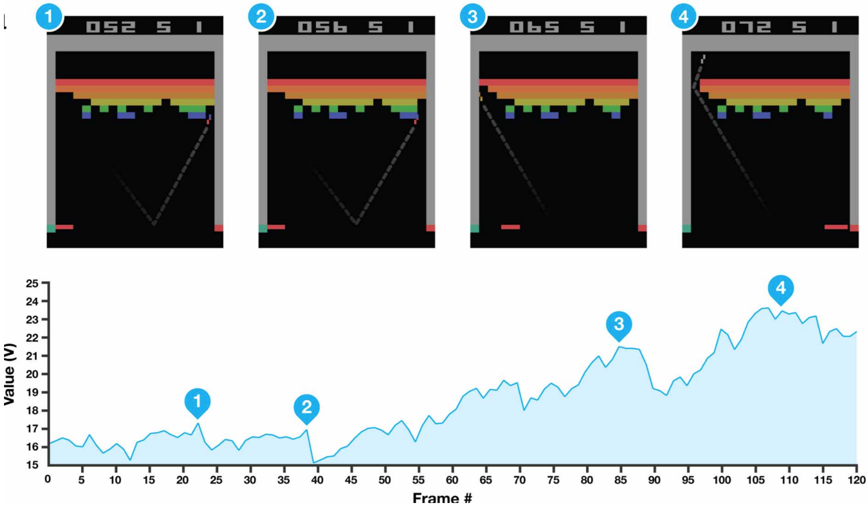


Reward vs computed Q-value



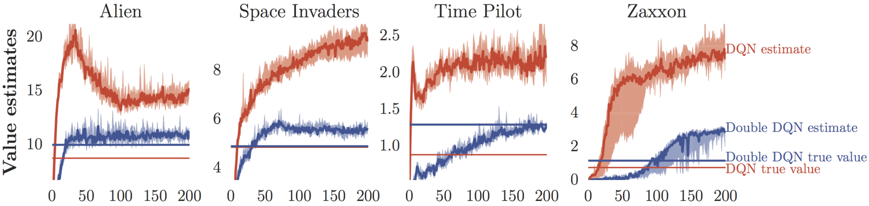
- The return increases as the predicted Q increases

Value vs behavior



- The value predicts the future reward

Does DQN actually find the true Q-value?



- DQN overestimates Q-value
  (DQN is very optimistic..)

Overestimation in Q-Learning

Target Q: $y_j = r_j + \gamma \max_a Q_{\phi^-}(s'_j, a)$;

Overestimation in Q-Learning

Target Q: $y_j = r_j + \gamma \max_a Q_{\phi^-}(s'_j, a)$;

Q) Why is the last term problematic?

Overestimation in Q-Learning

Target Q: $y_j = r_j + \gamma \max_a Q_{\phi^-}(s'_j, a)$;

Q) Why is the last term problematic?

- Given two random variables X_1 and X_2, we have $\mathbb{E}[\max\{X_1, X_2\}] \geq \max\{\mathbb{E}[X_1], \mathbb{E}[X_2]\}$

Overestimation in Q-Learning

Target Q: $y_j = r_j + \gamma \max_a Q_{\phi^-}(s'_j, a)$;

Q) Why is the last term problematic?

- Given two random variables X_1 and X_2, we have $\mathbb{E}[\max\{X_1, X_2\}] \geq \max\{\mathbb{E}[X_1], \mathbb{E}[X_2]\}$
- What we calculate is $\max_a Q_{\phi^-}(s', a) = \max_a X_a$, and estimates its mean $\mathbb{E}[\max_a X_a]$

Overestimation in Q-Learning

Target Q: $y_j = r_j + \gamma \max_a Q_{\phi^-}(s'_j, a)$;

Q) Why is the last term problematic?

- Given two random variables X_1 and X_2, we have $\mathbb{E}[\max\{X_1, X_2\}] \geq \max\{\mathbb{E}[X_1], \mathbb{E}[X_2]\}$
- What we calculate is $\max_a Q_{\phi^-}(s', a) = \max_a X_a$, and estimates its mean $\mathbb{E}[\max_a X_a]$
- It overestimates $\max_a \mathbb{E}[X_a] = \underbrace{\max_a \mathbb{E}[Q_{\phi^-}(s', a)]}_{\text{Want to compute}}$

Overestimation in Q-Learning

Target Q: $y_j = r_j + \gamma \max_a Q_{\phi^-}(s'_j, a)$;

Q) Why is the last term problematic?

- Given two random variables X_1 and X_2, we have $\mathbb{E}[\max\{X_1, X_2\}] \geq \max\{\mathbb{E}[X_1], \mathbb{E}[X_2]\}$
- What we calculate is $\max_a Q_{\phi^-}(s', a) = \max_a X_a$, and estimates its mean $\mathbb{E}[\max_a X_a]$
- It overestimates $\max_a \mathbb{E}[X_a] = \underbrace{\max_a \mathbb{E}[Q_{\phi^-}(s', a)]}_{\text{Want to compute}}$

$$\Longrightarrow \underbrace{\mathbb{E}\Big[\max_a Q_{\phi^-}(s', a)\Big]}_{\text{What we estimate}} \geq \underbrace{\max_a \mathbb{E}[Q_{\phi^-}(s', a)]}_{\text{Want to compute}}$$

Another look

What we calculate is

$$\max_a Q_{\phi^-}(s', a) = Q_{\phi^-}\left(s', \arg\max_a Q_{\phi^-}(s', a)\right)$$

Another look

What we calculate is

$$\max_a Q_{\phi^-}(s', a) = Q_{\phi^-}\left(s', \arg\max_a Q_{\phi^-}(s', a)\right)$$

Key observation:

- Action selected according to Q_{ϕ^-}
- Value also comes from Q_{ϕ^-}

Another look

What we calculate is

$$\max_a Q_{\phi^-}(s', a) = Q_{\phi^-}\left(s', \arg\max_a Q_{\phi^-}(s', a)\right)$$

Key observation:

- Action selected according to Q_{ϕ^-}
- Value also comes from Q_{ϕ^-}

Q) How can we suppress overestimation?

Another look

What we calculate is

$$\max_a Q_{\phi^-}(s', a) = Q_{\phi^-}\left(s', \arg\max_a Q_{\phi^-}(s', a)\right)$$

Key observation:

- Action selected according to Q_{ϕ^-}
- Value also comes from Q_{ϕ^-}

Q) How can we suppress overestimation?

- Make the noise in the two terms uncorrelated

Another look

What we calculate is

$$\max_a Q_{\phi^-}(s', a) = Q_{\phi^-}\left(s', \arg\max_a Q_{\phi^-}(s', a)\right)$$

Key observation:

- Action selected according to Q_{ϕ^-}
- Value also comes from Q_{ϕ^-}

Q) How can we suppress overestimation?

- Make the noise in the two terms uncorrelated

Q) How?

Another look

What we calculate is

$$\max_a Q_{\phi^-}(s', a) = Q_{\phi^-}\left(s', \arg\max_a Q_{\phi^-}(s', a)\right)$$

Key observation:

- Action selected according to Q_{ϕ^-}
- Value also comes from Q_{ϕ^-}

Q) How can we suppress overestimation?

- Make the noise in the two terms uncorrelated

Q) How?

- Use two Q-functions!

Double Q-Learning

Double Q-learning

Hado van Hasselt
Multi-agent and Adaptive Computation Group
Centrum Wiskunde & Informatica

Abstract

In some stochastic environments the well-known reinforcement learning algorithm Q-learning performs very poorly. This poor performance is caused by large overestimations of action values. These overestimations result from a positive bias that is introduced because Q-learning uses the maximum action value as an approximation for the maximum expected action value. We introduce an alternative way to approximate the maximum expected value for any set of random variables. The obtained double estimator method is shown to sometimes underestimate rather than overestimate the maximum expected value. We apply the double estimator to Q-learning to construct Double Q-learning, a new off-policy reinforcement learning algorithm. We show the new algorithm converges to the optimal policy and that it performs well in some settings in which Q-learning performs poorly due to its overestimation.

Idea: Use two Q-functions (two networks ϕ_A, ϕ_B):

Double Q-Learning

Double Q-learning

Hado van Hasselt
Multi-agent and Adaptive Computation Group
Centrum Wiskunde & Informatica

Abstract

In some stochastic environments the well-known reinforcement learning algorithm Q-learning performs very poorly. This poor performance is caused by large overestimations of action values. These overestimations result from a positive bias that is introduced because Q-learning uses the maximum action value as an approximation for the maximum expected action value. We introduce an alternative way to approximate the maximum expected value for any set of random variables. The obtained double estimator method is shown to sometimes underestimate rather than overestimate the maximum expected value. We apply the double estimator to Q-learning to construct Double Q-learning, a new off-policy reinforcement learning algorithm. We show the new algorithm converges to the optimal policy and that it performs well in some settings in which Q-learning performs poorly due to its overestimation.

Idea: Use two Q-functions (two networks ϕ_A, ϕ_B):

$$Q_{\phi_A}(s,a) \leftarrow r + \gamma Q_{\phi_B}\left(s', \arg\max_{a'} Q_{\phi_A}(s',a')\right)$$

$$Q_{\phi_B}(s,a) \leftarrow r + \gamma Q_{\phi_A}\left(s', \arg\max_{a'} Q_{\phi_B}(s',a')\right)$$

Double Q-Learning

Double Q-learning

Hado van Hasselt
Multi-agent and Adaptive Computation Group
Centrum Wiskunde & Informatica

Abstract

In some stochastic environments the well-known reinforcement learning algorithm Q-learning performs very poorly. This poor performance is caused by large overestimations of action values. These overestimations result from a positive bias that is introduced because Q-learning uses the maximum action value as an approximation for the maximum expected action value. We introduce an alternative way to approximate the maximum expected value for any set of random variables. The obtained double estimator method is shown to sometimes underestimate rather than overestimate the maximum expected value. We apply the double estimator to Q-learning to construct Double Q-learning, a new off-policy reinforcement learning algorithm. We show the new algorithm converges to the optimal policy and that it performs well in some settings in which Q-learning performs poorly due to its overestimation.

Idea: Use two Q-functions (two networks ϕ_A, ϕ_B):

$$Q_{\phi_A}(s,a) \leftarrow r + \gamma Q_{\phi_B}(s', \arg\max_{a'} Q_{\phi_A}(s',a'))$$

$$Q_{\phi_B}(s,a) \leftarrow r + \gamma Q_{\phi_A}(s', \arg\max_{a'} Q_{\phi_B}(s',a'))$$

- The two Q-functions are noisy in different ways!

Where to get two Q-functions?

Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Deep Reinforcement Learning with Double Q-Learning

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

- $Q_{\phi_A} := Q_\phi$
  Use current value to evaluate action
- $Q_{\phi_B} := Q_{\phi^-}$
  Use target network to compute value

Where to get two Q-functions?

Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Deep Reinforcement Learning with Double Q-Learning

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

- $Q_{\phi_A} := Q_\phi$
  Use current value to evaluate action
- $Q_{\phi_B} := Q_{\phi^-}$
  Use target network to compute value

Double DQN:

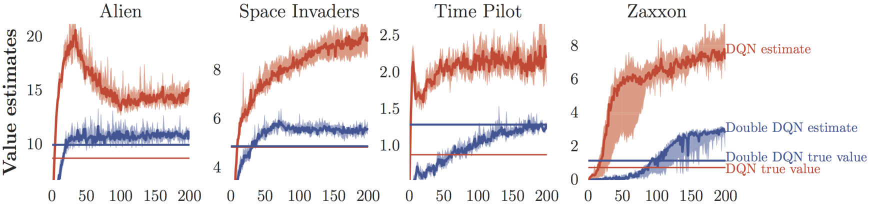
$$y \leftarrow r + \gamma Q_{\phi^-}\left(s', \arg\max_{a'} Q_\phi(s', a')\right)$$

Double DQN Algorithm

Initialize ϕ and ϕ^-;

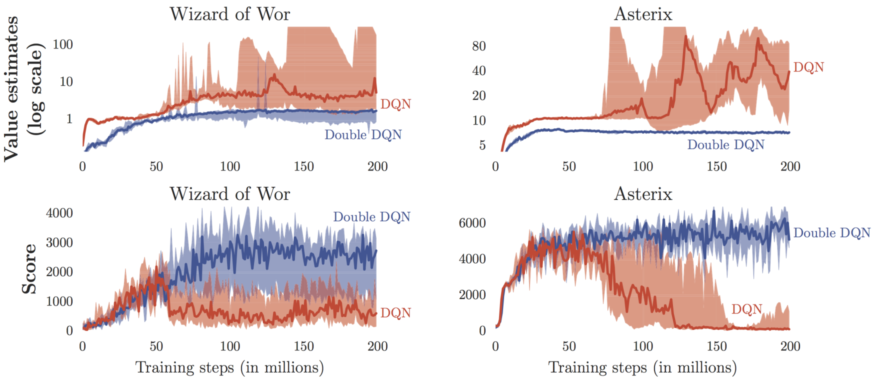
1. Save target network parameters: $\phi^- \leftarrow \phi$;
2. Repeat N times
   1. Collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add them to **Buffer**;
   2. Repeat K times
      1. Sample a mini-batch $\{(s_j, a_j, s'_j, r_j)\}$ from **Buffer**;
      2. Compute the target $y_j^- := r_j + \gamma \underbrace{Q_{\phi^-}\big(s'_j, \overbrace{\arg\max_{a'} Q_\phi(s'_j, a')}^{\text{action selection}}\big)}_{\text{value estimate}}$;
      3. Update $\phi \leftarrow \phi - \alpha \underbrace{\sum_j \frac{dQ_\phi}{d\phi}(Q_\phi(s_j, a_j) - y_j^-)}_{\text{stochastic gradient}}$;
3. Repeat until converges;

Result I



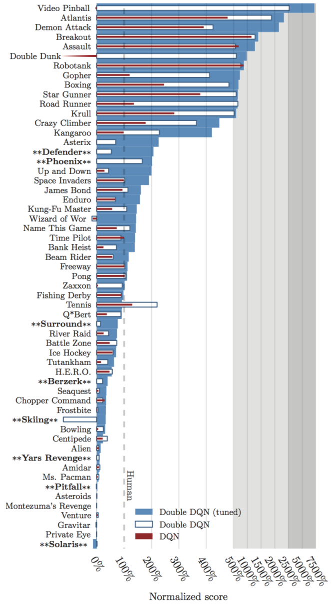
- Double DQN more accurately estimates Q-value than DQN

Result II



- Double DQN performs better than DQN
- Double DQN training is more stable than DQN

Result III



Video Pinball
Atlantis
Demon Attack
Breakout
Assault
Double Dunk
Robotank
Gopher
Boxing
Star Gunner
Road Runner
Krull
Crazy Climber
Kangaroo
Asterix
Defender
Phoenix
Up and Down
Space Invaders
James Bond
Enduro
Kung-Fu Master
Wizard of Wor
Name This Game
Time Pilot
Bank Heist
Beam Rider
Freeway
Pong
Zaxxon
Fishing Derby
Tennis
Q*Bert
Surround
River Raid
Battle Zone
Ice Hockey
Tutankham
H.E.R.O.
Berzerk
Seaquest
Chopper Command
Frostbite
Skiing
Bowling
Centipede
Alien
Yars Revenge
Amidar
Ms. Pacman
Pitfall
Asteroids
Montezuma's Revenge
Venture
Gravitar
Private Eye
Solaris
Human
Double DQN (tuned)
Double DQN
DQN
0%
100%
200%
300%
400%
500%
1000%
1500%
2000%
2500%
5000%
7500%
Normalized score

Advantages and Disadvantages

Advantages:

- Very simple
- Sample efficient
- Fairly stable training

Advantages and Disadvantages

Advantages:

- Very simple
- Sample efficient
- Fairly stable training

Disadvantages:

- Continuous control
- Exploration
- Complex tasks

Today's Review

- DQN = Q-learning + deep learning
  1. Experience replay
  2. Slow update of target network

Today's Review

- DQN = Q-learning + deep learning
  1. Experience replay
  2. Slow update of target network

- Double DQN = DQN + double Q-learning trick
  1. Reduce overestimation
  2. Very simple modification:
     Q_ϕ for action selection
     Q_{ϕ^-} for value estimate