ECEN 743: Reinforcement Learning

Deep Q-Learning

Dileep Kalathil
Assistant Professor
Department of Electrical and Computer Engineering
Texas A&M University

Experience Replay and Target Network

Correlation and Forgetting in Q-Learning

- Q-learning with function approximation

Correlation and Forgetting in Q-Learning

- Q-learning with function approximation
  1. Take action a_t and get the data point $e_t = (s_t, a_t, r_t, s_{t+1})$

Correlation and Forgetting in Q-Learning

- Q-learning with function approximation
  1. Take action a_t and get the data point $e_t = (s_t, a_t, r_t, s_{t+1})$
  2. Update $w_{t+1} = w_t + \alpha_t \left(r_t + \gamma \max_b Q(w_t)(s_{t+1}, b) - Q(w_t)(s_t, a_t)\right) \nabla_w Q(w_t)(s_t, a_t)$

Correlation and Forgetting in Q-Learning

- Q-learning with function approximation
  1. Take action a_t and get the data point $e_t = (s_t, a_t, r_t, s_{t+1})$
  2. Update $w_{t+1} = w_t + \alpha_t \left(r_t + \gamma \max_b Q(w_t)(s_{t+1}, b) - Q(w_t)(s_t, a_t)\right) \nabla_w Q(w_t)(s_t, a_t)$
- Data is used sequentially, in the same way as they are observed

Correlation and Forgetting in Q-Learning

- Q-learning with function approximation
  1. Take action a_t and get the data point $e_t = (s_t, a_t, r_t, s_{t+1})$
  2. Update $w_{t+1} = w_t + \alpha_t \left(r_t + \gamma \max_b Q(w_t)(s_{t+1}, b) - Q(w_t)(s_t, a_t)\right) \nabla_w Q(w_t)(s_t, a_t)$
- Data is used sequentially, in the same way as they are observed
- Past data is discarded

Correlation and Forgetting in Q-Learning

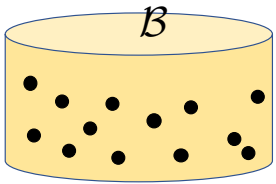
- Q-learning with function approximation
  1. Take action a_t and get the data point $e_t = (s_t, a_t, r_t, s_{t+1})$
  2. Update $w_{t+1} = w_t + \alpha_t \left(r_t + \gamma \max_b Q(w_t)(s_{t+1}, b) - Q(w_t)(s_t, a_t)\right) \nabla_w Q(w_t)(s_t, a_t)$
- Data is used sequentially, in the same way as they are observed
- Past data is discarded
- Correlation: Sequential data leads to correlated updates. This breaks the i.i.d. assumption of many popular SGD algorithms

Correlation and Forgetting in Q-Learning

- Q-learning with function approximation
  1. Take action a_t and get the data point $e_t = (s_t, a_t, r_t, s_{t+1})$
  2. Update $w_{t+1} = w_t + \alpha_t \left(r_t + \gamma \max_b Q(w_t)(s_{t+1}, b) - Q(w_t)(s_t, a_t)\right) \nabla_w Q(w_t)(s_t, a_t)$
- Data is used sequentially, in the same way as they are observed
- Past data is discarded
- Correlation: Sequential data leads to correlated updates. This breaks the i.i.d. assumption of many popular SGD algorithms
- Rapid forgetting: Information from not-so-frequent experiences (that would be useful later on) are not effectively used

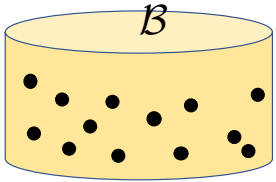
Experience Relay

- Store samples of prior experience in replay buffer
- Sample from the replay buffer randomly for updating Q-value



Experience Relay

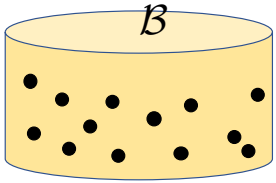
- Store samples of prior experience in replay buffer
- Sample from the replay buffer randomly for updating Q-value



1. Initialization: Collect data $\{e_j = (s_j, a_j, r_j, s_{j+1})\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.

Experience Relay

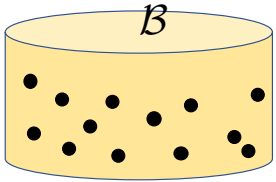
- Store samples of prior experience in replay buffer
- Sample from the replay buffer randomly for updating Q-value



1. Initialization: Collect data $\{e_j = (s_j, a_j, r_j, s_{j+1})\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.
2. At each time t, get a random sample (or a mini-batch of random samples) e_k from $\mathcal{B}$

Experience Relay

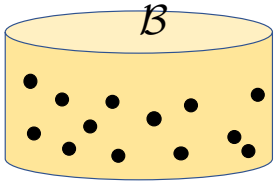
- Store samples of prior experience in replay buffer
- Sample from the replay buffer randomly for updating Q-value



1. Initialization: Collect data $\{e_j = (s_j, a_j, r_j, s_{j+1})\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.
2. At each time t, get a random sample (or a mini-batch of random samples) e_k from $\mathcal{B}$
3. For each sampled e_k, update $w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$

Experience Relay

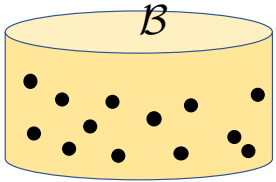
- Store samples of prior experience in replay buffer
- Sample from the replay buffer randomly for updating Q-value



1. Initialization: Collect data $\{e_j = (s_j, a_j, r_j, s_{j+1})\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.
2. At each time t, get a random sample (or a mini-batch of random samples) e_k from $\mathcal{B}$
3. For each sampled e_k, update $w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$
4. Select an action a_t according to ϵ-greedy policy

Experience Relay

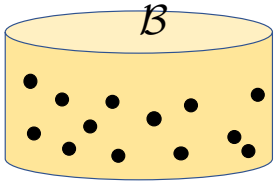
- Store samples of prior experience in replay buffer
- Sample from the replay buffer randomly for updating Q-value



1. Initialization: Collect data $\{e_j = (s_j, a_j, r_j, s_{j+1})\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.
2. At each time t, get a random sample (or a mini-batch of random samples) e_k from $\mathcal{B}$
3. For each sampled e_k, update $w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$
4. Select an action a_t according to ϵ-greedy policy
5. Add the new data point $e_t = (s_t, a_t, r_t, s'_t)$ to $\mathcal{B}$.

Experience Relay

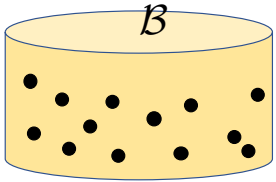
- Store samples of prior experience in replay buffer
- Sample from the replay buffer randomly for updating Q-value



1. Initialization: Collect data $\{e_j = (s_j, a_j, r_j, s_{j+1})\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.
2. At each time t, get a random sample (or a mini-batch of random samples) e_k from $\mathcal{B}$
3. For each sampled e_k, update $w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$
4. Select an action a_t according to ϵ-greedy policy
5. Add the new data point $e_t = (s_t, a_t, r_t, s'_t)$ to $\mathcal{B}$.

Experience Relay

- Store samples of prior experience in replay buffer
- Sample from the replay buffer randomly for updating Q-value



1. Initialization: Collect data $\{e_j = (s_j, a_j, r_j, s_{j+1})\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.
2. At each time t, get a random sample (or a mini-batch of random samples) e_k from $\mathcal{B}$
3. For each sampled e_k, update $w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$
4. Select an action a_t according to ϵ-greedy policy
5. Add the new data point $e_t = (s_t, a_t, r_t, s'_t)$ to $\mathcal{B}$.

- Experience replay breaks the temporal correlations by mixing data
- Experience replay uses (rare) experiences for more than just a single update

Target Network

- Q-learning gradient update:

$$w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k) \right) \nabla_w Q(w)(s_k, a_k)$$

Target Network

- Q-learning gradient update:

$$w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$$

- In supervised learning, the target does not depend on the parameter

Target Network

- Q-learning gradient update:

$$w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k) \right) \nabla_w Q(w)(s_k, a_k)$$

- In supervised learning, the target does not depend on the parameter
- In Q-learning, target $r_k + \gamma \max_b Q(w)(s_{k+1}, b)$ depends on w

Target Network

- Q-learning gradient update:

$$w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$$

- In supervised learning, the target does not depend on the parameter
- In Q-learning, target $r_k + \gamma \max_b Q(w)(s_{k+1}, b)$ depends on w
  - Target is moving as the parameter is updated

Target Network

- Q-learning gradient update:

$$w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$$

- In supervised learning, the target does not depend on the parameter
- In Q-learning, target $r_k + \gamma \max_b Q(w)(s_{k+1}, b)$ depends on w
  - Target is moving as the parameter is updated
  - This leads to oscillation and instability

Target Network

- Q-learning gradient update:

$$w = w + \alpha \left(r_k + \gamma \max_b Q(w)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$$

- In supervised learning, the target does not depend on the parameter
- In Q-learning, target $r_k + \gamma \max_b Q(w)(s_{k+1}, b)$ depends on w
  - Target is moving as the parameter is updated
  - This leads to oscillation and instability
- Target network: A separate target network and keep it unchanged for multiple updates

$$w = w + \alpha \left(r_k + \gamma \max_b Q(w^-)(s_{k+1}, b) - Q(w)(s_k, a_k)\right) \nabla_w Q(w)(s_k, a_k)$$

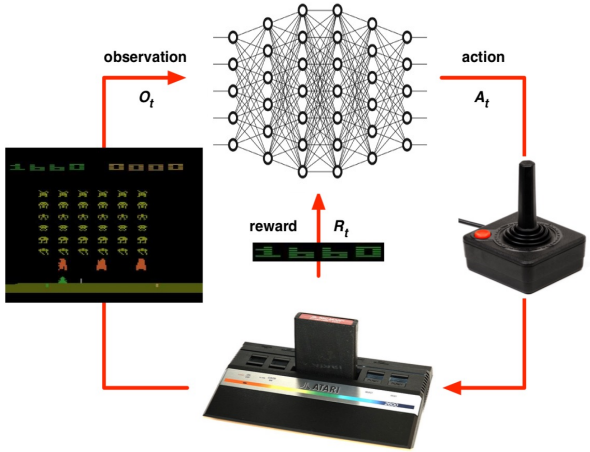
$w^- = w$, after every N steps

Deep Q Networks

Mnih et al, "Human-level control through deep reinforcement learning", *Nature*, 2015

Playing Atari Games using RL

- States: screen Images, 210 x 160 pixel, 128 color
- Actions: joystick positions, 18 different positions
- Reward: Game score



- Objective: Learn to win the game (a control policy that maximizes game score)

DQN for Atari

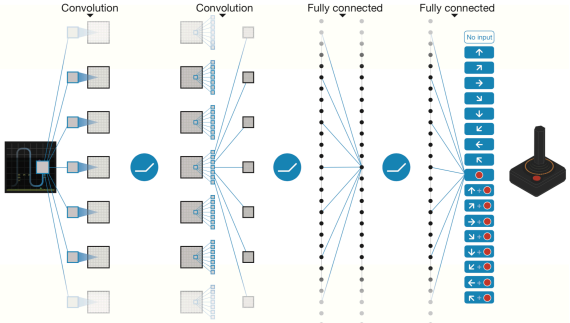
- Pre-processing
  - Each original frame is 210 x 160 pixel images with a 128-colour palette
  - Reduce it to to 84 x 84 images by pre-processing
  - Use the 4 most recent frame
  - Dimension of the state 84 x 84 x 4

DQN for Atari

- Pre-processing
  - Each original frame is 210 x 160 pixel images with a 128-colour palette
  - Reduce it to to 84 x 84 images by pre-processing
  - Use the 4 most recent frame
  - Dimension of the state 84 x 84 x 4
- Architecture
  - Input state s, output $Q(s, a)$, for each a
  - Three CNN and two fully connected layers with ReLU

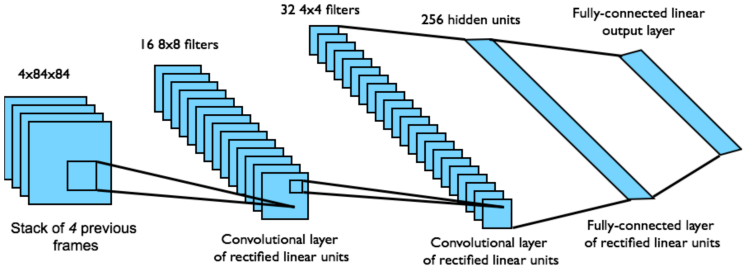
DQN for Atari

- Pre-processing
  - Each original frame is 210 x 160 pixel images with a 128-colour palette
  - Reduce it to to 84 x 84 images by pre-processing
  - Use the 4 most recent frame
  - Dimension of the state 84 x 84 x 4
- Architecture
  - Input state s, output $Q(s, a)$, for each a
  - Three CNN and two fully connected layers with ReLU



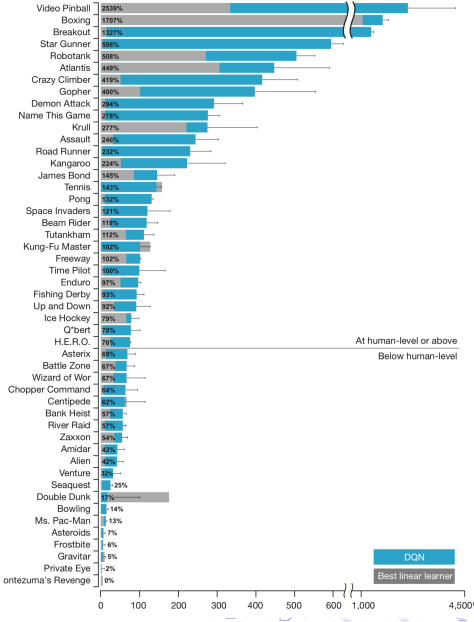
DQN for Atari

- Pre-processing
  - Each original frame is 210 x 160 pixel images with a 128-colour palette
  - Reduce it to to 84 x 84 images by pre-processing
  - Use the 4 most recent frame
  - Dimension of the state 84 x 84 x 4
- Architecture
  - Input state s, output $Q(s, a)$, for each a
  - Three CNN and two fully connected layers with ReLU



DQN Performance in Atari

- 100 x (DQN score - random play score)/(human score - random play score)
- Score for each game is averaged over 30 sessions on each game
- Professional human tester played using the same emulator
- DQN played better than the human player on 22 of the games
- DQN played better than the 75% of the human player performance on 29 of the games



DQN Training

- Training time: Training over 50 million frames. 38 days of game experience in total
- Replay memory: Recent 1 million frames
- Minibatch size: 32
- Target network update frequency: After every 10k parameter updates
- Action repeat: Repeat the same action for k (= 4) frames
- SGD: RMSProp, with learning rate 0.00025
- Exploration: Epsilon-greedy policy, with epsilon decreasing from 1.0 to 0.1 over first million frames and then fixed after
- Discount factor: 0.99

Effect of Replay and Target Network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

DQN Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

DQN Improvements

- Double DQN: [Van Hasselt et al., 2016][1], [Van Hasselt, 2010][2]
- Prioritized Experience Replay: [Schaul et al, 2016] [3]
- Rainbow DQN: [Hessel et al, 2018][4]

[1] H Van Hasselt, A Guez, D Silver, "Deep Reinforcement Learning with Double Q-Learning", *AAAI*, 2016
[2] Van Hasselt, "Double Q-Learning", *NeurIPS*, 2010
[3] Schaul et al, "Prioritized experience replay", *ICLR*, 2016
[4] Hessel et al, "Rainbow: Combining Improvements in Deep Reinforcement Learning", *AAAI*, 2018

DQN Improvements:
Double DQN

- Double DQN: [Van Hasselt et al., 2016][5], [Van Hasselt, 2010][6]

[5] H Van Hasselt, A Guez, D Silver, "Deep Reinforcement Learning with Double Q-Learning", *AAAI*, 2016
[6] Van Hasselt, "Double Q-Learning", *NeurIPS*, 2010

Maximization Bias in Q-Learning

- Let $\widehat{\pi} = \arg\max_a \widehat{Q}(s,a)$ and $\widehat{V}_{\widehat{\pi}} = \max_a \widehat{Q}(s,a)$

Maximization Bias in Q-Learning

- Let $\widehat{\pi} = \arg\max_a \widehat{Q}(s,a)$ and $\widehat{V}_{\widehat{\pi}} = \max_a \widehat{Q}(s,a)$

$$\mathbb{E}[\widehat{V}_{\widehat{\pi}}] = \mathbb{E}[\max_a \widehat{Q}(s,a)] \geq \max_a \mathbb{E}[\widehat{Q}(s,a)]$$

Maximization Bias in Q-Learning

- Let $\widehat{\pi} = \arg\max_a \widehat{Q}(s,a)$ and $\widehat{V}_{\widehat{\pi}} = \max_a \widehat{Q}(s,a)$

$$\mathbb{E}[\widehat{V}_{\widehat{\pi}}] = \mathbb{E}[\max_a \widehat{Q}(s,a)] \geq \max_a \mathbb{E}[\widehat{Q}(s,a)]$$

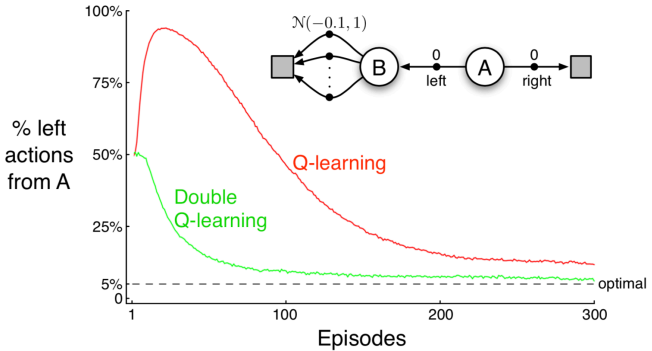
- The greedy policy obtained from the estimated Q values can yield a maximization bias when the number of samples are finite

Maximization Bias in Q-Learning

- Let $\widehat{\pi} = \arg\max_a \widehat{Q}(s,a)$ and $\widehat{V}_{\widehat{\pi}} = \max_a \widehat{Q}(s,a)$

$$\mathbb{E}[\widehat{V}_{\widehat{\pi}}] = \mathbb{E}[\max_a \widehat{Q}(s,a)] \geq \max_a \mathbb{E}[\widehat{Q}(s,a)]$$

- The greedy policy obtained from the estimated Q values can yield a maximization bias when the number of samples are finite
- Example 6.7, Sutton and Barto



Maximization Bias in Q-Learning

- Let $\widehat{\pi} = \arg\max_a \widehat{Q}(s,a)$ and $\widehat{V}_{\widehat{\pi}} = \max_a \widehat{Q}(s,a)$

$$\mathbb{E}[\widehat{V}_{\widehat{\pi}}] = \mathbb{E}[\max_a \widehat{Q}(s,a)] \geq \max_a \mathbb{E}[\widehat{Q}(s,a)]$$

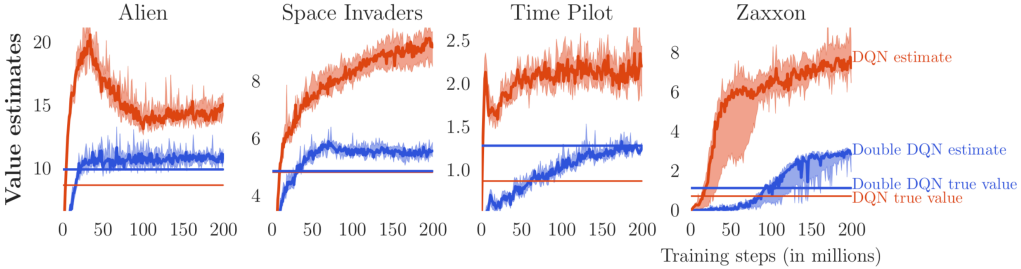
- The greedy policy obtained from the estimated Q values can yield a maximization bias when the number of samples are finite

Maximization Bias in Q-Learning

- Let $\widehat{\pi} = \arg\max_a \widehat{Q}(s,a)$ and $\widehat{V}_{\widehat{\pi}} = \max_a \widehat{Q}(s,a)$

$$\mathbb{E}[\widehat{V}_{\widehat{\pi}}] = \mathbb{E}[\max_a \widehat{Q}(s,a)] \geq \max_a \mathbb{E}[\widehat{Q}(s,a)]$$

- The greedy policy obtained from the estimated Q values can yield a maximization bias when the number of samples are finite
- Maximization bias is DQN, from [Van Hasselt et al., 2016]



Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.
  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.

  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$
  - Single estimator gives an overestimate

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.

  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$
  - Single estimator gives an overestimate
  - Using the same samples both to determine the maximizer and value

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.

  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$
  - Single estimator gives an overestimate
  - Using the same samples both to determine the maximizer and value
- Possible solution: divide the samples into two sets and use them to learn two independent estimates

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.

  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$
  - Single estimator gives an overestimate
  - Using the same samples both to determine the maximizer and value
- Possible solution: divide the samples into two sets and use them to learn two independent estimates
- Double estimator:

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.

  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$
  - Single estimator gives an overestimate
  - Using the same samples both to determine the maximizer and value
- Possible solution: divide the samples into two sets and use them to learn two independent estimates
- Double estimator:
  1. Split $\mathcal{Y}^i$ into $\mathcal{Y}^i_A$ and $\mathcal{Y}^i_B$

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.

  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$
  - Single estimator gives an overestimate
  - Using the same samples both to determine the maximizer and value
- Possible solution: divide the samples into two sets and use them to learn two independent estimates
- Double estimator:
  1. Split $\mathcal{Y}^i$ into $\mathcal{Y}^i_A$ and $\mathcal{Y}^i_B$
  2. Compute $\hat{\mu}^i_A = \frac{1}{|\mathcal{Y}^i_A|} \sum_{y \in \mathcal{Y}^i_A} y$, $\hat{\mu}^i_B = \frac{1}{|\mathcal{Y}^i_B|} \sum_{y \in \mathcal{Y}^i_B} y$

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.

  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$
  - Single estimator gives an overestimate
  - Using the same samples both to determine the maximizer and value
- Possible solution: divide the samples into two sets and use them to learn two independent estimates
- Double estimator:
  1. Split $\mathcal{Y}^i$ into $\mathcal{Y}^i_A$ and $\mathcal{Y}^i_B$
  2. Compute $\hat{\mu}^i_A = \frac{1}{|\mathcal{Y}^i_A|} \sum_{y \in \mathcal{Y}^i_A} y$, $\hat{\mu}^i_B = \frac{1}{|\mathcal{Y}^i_B|} \sum_{y \in \mathcal{Y}^i_B} y$
  3. Find $\hat{i}^* = \arg\max_i \hat{\mu}^i_A$ and get double estimator $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}_B$

Double Estimator

- Consider two random variables Y^1 and Y^2 with mean values μ^1 and μ^2
- Let $\mathcal{Y}^1$ and $\mathcal{Y}^2$ be the sets of i.i.d. samples of Y^1 and Y^2
- Our goal is to estimate $\max_i \mu^i$
- Single estimator: Let $\hat{\mu}^i = \frac{1}{|\mathcal{Y}^i|} \sum_{y \in \mathcal{Y}^i} y$. Let $\hat{i}^* = \arg\max_i \hat{\mu}^i$.

  Then, single estimator is $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}$
  - Single estimator gives an overestimate
  - Using the same samples both to determine the maximizer and value
- Possible solution: divide the samples into two sets and use them to learn two independent estimates
- Double estimator:
  1. Split $\mathcal{Y}^i$ into $\mathcal{Y}^i_A$ and $\mathcal{Y}^i_B$
  2. Compute $\hat{\mu}^i_A = \frac{1}{|\mathcal{Y}^i_A|} \sum_{y \in \mathcal{Y}^i_A} y$, $\hat{\mu}^i_B = \frac{1}{|\mathcal{Y}^i_B|} \sum_{y \in \mathcal{Y}^i_B} y$
  3. Find $\hat{i}^* = \arg\max_i \hat{\mu}^i_A$ and get double estimator $\hat{\mu}_{\max} = \hat{\mu}^{\hat{i}^*}_B$
- We can show that the double estimator is an underestimator, i.e., $\mathbb{E}[\hat{\mu}_{\max}] \leq \max_i \mathbb{E}[Y^i]$

Double Estimator

Proof: Let $i^* = \arg\max_i \mathbb{E}[Y^i]$ be the true maximizer. Then,

$$\begin{aligned}
\mathbb{E}[\hat{\mu}_{\max}] &= \mathbb{E}[\hat{\mu}_B^{\hat{i}^*}] = \mathbb{P}(\hat{i}^* = i^*)\mathbb{E}[\hat{\mu}_B^{\hat{i}^*}|\hat{i}^* = i^*] + \mathbb{P}(\hat{i}^* \neq i^*)\mathbb{E}[\hat{\mu}_B^{\hat{i}^*}|\hat{i}^* \neq i^*] \\
&= \mathbb{P}(\hat{i}^* = i^*) \max_i \mathbb{E}[Y^i] + \mathbb{P}(\hat{i}^* \neq i^*)\mathbb{E}[\hat{\mu}_B^{\hat{i}^*}|\hat{i}^* \neq i^*] \\
&\leq \mathbb{P}(\hat{i}^* = i^*) \max_i \mathbb{E}[Y^i] + \mathbb{P}(\hat{i}^* \neq i^*) \max_i \mathbb{E}[Y^i] = \max_i \mathbb{E}[Y^i]
\end{aligned}$$

□

Double DQN

- Use one Q-network to select the max-action and another Q-network to evaluate it

Double DQN

- Use one Q-network to select the max-action and another Q-network to evaluate it
- DQN maintains two networks anyway (one as the target network and one as the current network)

$$w = w + \alpha \left(R + \gamma \max_b Q(w^-)(s', b) - Q(w)(s, a)\right) \nabla Q(w)(s, a)$$

$$w^- = w, \text{ after every } N \text{ steps}$$

Double DQN

- Use one Q-network to select the max-action and another Q-network to evaluate it
- DQN maintains two networks anyway (one as the target network and one as the current network)

$$w = w + \alpha \left(R + \gamma \max_b Q(w^-)(s', b) - Q(w)(s, a)\right) \nabla Q(w)(s, a)$$

$$w^- = w, \text{ after every } N \text{ steps}$$

- To implement, double DQN, use the current Q-network to select the max-action, and the target Q-network to evaluate that action

Double DQN

- Use one Q-network to select the max-action and another Q-network to evaluate it
- DQN maintains two networks anyway (one as the target network and one as the current network)

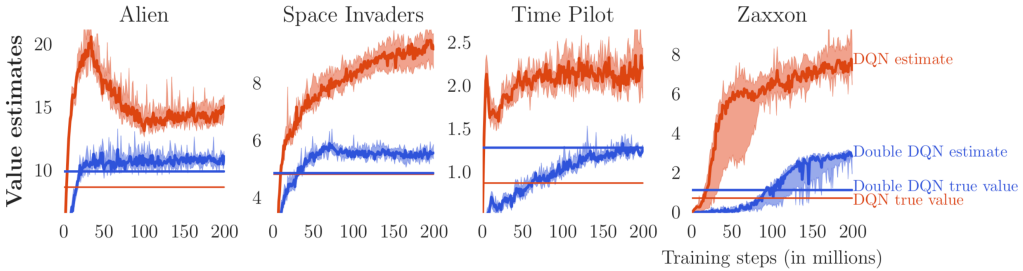
$$w = w + \alpha \left(R + \gamma \max_b Q(w^-)(s', b) - Q(w)(s, a)\right) \nabla Q(w)(s, a)$$

$$w^- = w, \text{ after every } N \text{ steps}$$

- To implement, double DQN, use the current Q-network to select the max-action, and the target Q-network to evaluate that action

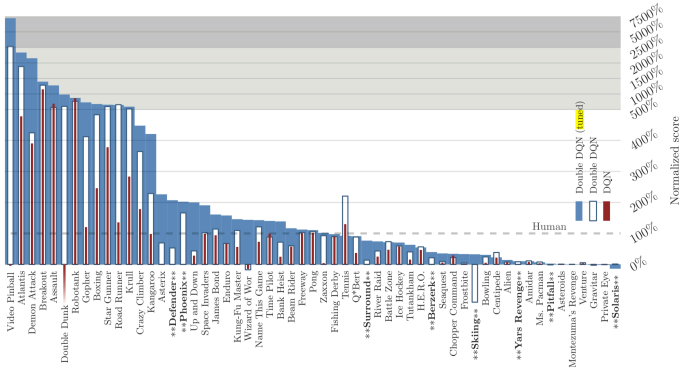
$$w = w + \alpha \left(r + \gamma Q(w^-)(s', \arg\max_b Q(w)(s', b)) - Q(w)(s, a)\right) \nabla Q(w)(s, a)$$

Double DQN Performance



Images from [Van Hasselt et al., 2016]

Double DQN Performance



	no ops		human starts		
	DQN	DDQN	DQN	DDQN	DDQN (tuned)
Median	93%	**115**%	47%	88%	**117**%
Mean	241%	**330**%	122%	273%	**475**%

DQN Improvements
Prioritized Experience Replay

- Prioritized Experience Replay: [Schaul et al, 2016] [9]

[9] Schaul et al, "Prioritized experience replay", *ICLR*, 2016

Sampling from Replay Buffer

- Some experience (data samples (s_i, a_i, r_i, s_{i+1})) may be more informative than other

Sampling from Replay Buffer

- Some experience (data samples (s_i, a_i, r_i, s_{i+1})) may be more informative than other
- In experience replay, samples are taken uniformly at random from the replay buffer

Sampling from Replay Buffer

- Some experience (data samples (s_i, a_i, r_i, s_{i+1})) may be more informative than other
- In experience replay, samples are taken uniformly at random from the replay buffer
- Uniform sampling will not be able to make use of such informative samples effectively

Sampling from Replay Buffer

- Some experience (data samples (s_i, a_i, r_i, s_{i+1})) may be more informative than other
- In experience replay, samples are taken uniformly at random from the replay buffer
- Uniform sampling will not be able to make use of such informative samples effectively
- Can we prioritize the samples to accelerate the learning progress?

Prioritized Experience Replay

- Intuition: Prioritize a sample based on how much can learn from a transition (expected learning progress)

Image from [Schaul et al, 2016]

Prioritized Experience Replay

- Intuition: Prioritize a sample based on how much can learn from a transition (expected learning progress)
- Proxy for this is TD error: For $e_i = (s_i, a_i, r_i, s_{i+1})$, TD error is defined as $\delta_i = r_i + \gamma \max_b Q(w)(s_{i+1}, b) - Q(w)(s_i, a_i)$

Image from [Schaul et al, 2016]

Prioritized Experience Replay

- Intuition: Prioritize a sample based on how much can learn from a transition (expected learning progress)
- Proxy for this is TD error: For $e_i = (s_i, a_i, r_i, s_{i+1})$, TD error is defined as $\delta_i = r_i + \gamma \max_b Q(w)(s_{i+1}, b) - Q(w)(s_i, a_i)$
- In prioritized experience replay, sample e_i is selected with probability p_i,

Image from [Schaul et al, 2016]

Prioritized Experience Replay

- Intuition: Prioritize a sample based on how much can learn from a transition (expected learning progress)
- Proxy for this is TD error: For $e_i = (s_i, a_i, r_i, s_{i+1})$, TD error is defined as $\delta_i = r_i + \gamma \max_b Q(w)(s_{i+1}, b) - Q(w)(s_i, a_i)$
- In prioritized experience replay, sample e_i is selected with probability p_i,
  - Proportional sampling: $p_i = \frac{|\delta_i|^c}{\sum_i |\delta_i|^c}$

Image from [Schaul et al, 2016]

Prioritized Experience Replay

- Intuition: Prioritize a sample based on how much can learn from a transition (expected learning progress)
- Proxy for this is TD error: For $e_i = (s_i, a_i, r_i, s_{i+1})$, TD error is defined as $\delta_i = r_i + \gamma \max_b Q(w)(s_{i+1}, b) - Q(w)(s_i, a_i)$
- In prioritized experience replay, sample e_i is selected with probability p_i,
  - Proportional sampling: $p_i = \frac{|\delta_i|^c}{\sum_i |\delta_i|^c}$
  - Rank-based sampling: $p_i = \frac{1}{\mathsf{rank}(e_i)}$, where the rank of e_i when the replay memory is sorted according to $|\delta_i|$

Image from [Schaul et al, 2016]

Prioritized Experience Replay

- Intuition: Prioritize a sample based on how much can learn from a transition (expected learning progress)
- Proxy for this is TD error: For $e_i = (s_i, a_i, r_i, s_{i+1})$, TD error is defined as $\delta_i = r_i + \gamma \max_b Q(w)(s_{i+1}, b) - Q(w)(s_i, a_i)$
- In prioritized experience replay, sample e_i is selected with probability p_i,
  - Proportional sampling: $p_i = \frac{|\delta_i|^c}{\sum_i |\delta_i|^c}$
  - Rank-based sampling: $p_i = \frac{1}{\mathsf{rank}(e_i)}$, where the rank of e_i when the replay memory is sorted according to $|\delta_i|$
- Prioritized experience replay: performance improvement

	DQN		Double DQN (tuned)		
	baseline	rank-based	baseline	rank-based	proportional
Median	48%	106%	111%	113%	128%
Mean	122%	355%	418%	454%	551%
> baseline	–	41	–	38	42
> human	15	25	30	33	33
# games	49	49	57	57	57

Image from [Schaul et al, 2016]

DQN Improvements:
Rainbow DQN

- Rainbow DQN: [Hessel et al., 2018][11]

[11] Hessel et al, "Rainbow: Combining Improvements in Deep Reinforcement Learning", *AAAI*, 2018

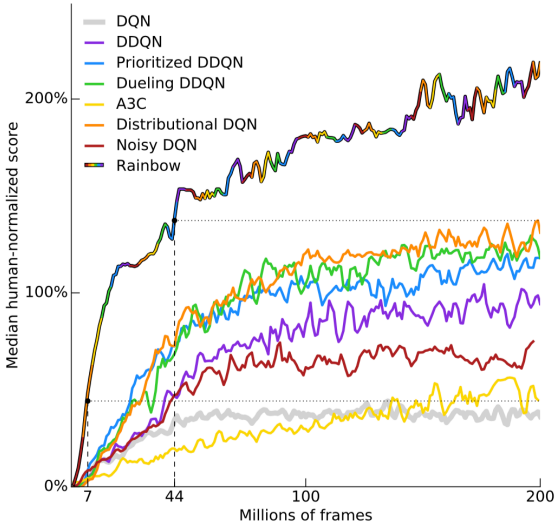
Rainbow DQN

Abstract

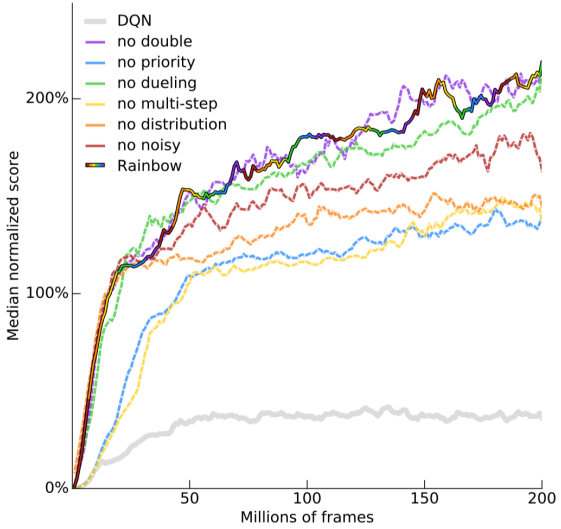
The deep reinforcement learning community has made several independent improvements to the DQN algorithm. However, it is unclear which of these extensions are complementary and can be fruitfully combined. This paper examines six extensions to the DQN algorithm and empirically studies their combination. 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.

Image from [Hessel et al., 2018]

Rainbow DQN: Performance



Rainbow DQN: Ablation Studies



Agent57

- Agent57: Outperforming the human Atari benchmark [Link]
  - "We have developed Agent57, the first deep reinforcement learning agent to obtain a score that is above the human baseline on all 57 Atari 2600 games"