Lecture 6: Model-free RL with Value Function Approximation Continued [1]

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2023

[1] With some slides based on slides for DQN from David Silver

Class Structure

- Last time: Model-free value function approximation control and Deep Q-learning
- This time: Model-free value function approximation and more DQN
- Next time: Policy search in large spaces / policy gradient methods

Refresh Your Understanding: Modified AB Example: (Ex. 6.4, Sutton & Barto, 2018)

TD → certainty equivalent MDP soln for V^π

- Two states A, B with $\gamma = 1$. Policy evaluation.
- Given 8 episodes of experience:
  - $A, 1, B, 0$ (observed 2 times) (state, reward, next state, next reward)
  - $B, 1$ (observed 4 times) (state, reward)
  - $B, 0$ (observed 2 times) (state, reward)
- Imagine run TD updates over data infinite number of times, and (separately) MC over data an infinite number of times?
- What is $V^{TD}(B)$ and $V^{TD}(A)$? What is $V^{MC}(B)$ and $V^{MC}(A)$?

.5   1.5   .5   1

$r + \gamma V(B)$

$= 1 + \gamma \cdot .5 = 1.5$

$1 + 0 = 1$

$1 + 0 = 1$

Refresh Your Understanding: Modified AB Example: (Ex. 6.4, Sutton & Barto, 2018). Solution

- Two states A, B with $\gamma = 1$
- Given 8 episodes of experience:
  - $A, 1, B, 0$ (observed 2 times)
  - $B, 1$ (observed 4 times)
  - $B, 0$ (observed 2 times)
- Imagine run TD updates over data infinite number of times, and (separately) MC over data an infinite number of times?
- What is $V^{TD}(B)$ and $V^{TD}(A)$? What is $V^{MC}(B)$ and $V^{MC}(A)$? V(B) = 0.5 for TD and MC. V(A) = 1.5 for TD. V(A) = 1.0 for MC.

Table of Contents

Table of Contents

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation

- Define the mean squared error of a linear value function approximation for a particular policy π relative to the true value as

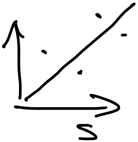
$$MSVE_{\mu}(\boldsymbol{w}) = \sum_{s \in S} \mu(s)(V^{\pi}(s) - \hat{V}^{\pi}(s; \boldsymbol{w}))^2$$

- where
  - $\mu(s)$: probability of visiting state s under policy π . Note $\sum_s \mu(s) = 1$
  - $\hat{V}^{\pi}(s; \boldsymbol{w}) = \boldsymbol{x}(s)^T \boldsymbol{w}$, a linear value function approximation

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation

- Define the mean squared error of a linear value function approximation for a particular policy π relative to the true value as

$$MSVE_{\mu}(\mathbf{w}) = \sum_{s \in S} \mu(s)(V^{\pi}(s) - \hat{V}^{\pi}(s; \mathbf{w}))^2$$



- where
  - $\mu(s)$: probability of visiting state s under policy π . Note $\sum_s \mu(s) = 1$
  - $\hat{V}^{\pi}(s; \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w}$, a linear value function approximation
- Monte Carlo policy evaluation with VFA converges to the weights $\mathbf{w}_{MC}$ which has the minimum mean squared error possible with respect to the distribution μ:

no Markov assumptions

$$MSVE_{\mu}(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_{s \in S} \mu(s)(V^{\pi}(s) - \hat{V}^{\pi}(s; \mathbf{w}))^2$$

Convergence Guarantees for TD Linear VFA for Policy Evaluation: Preliminaries

- For infinite horizon, the Markov Chain defined by a MDP with a particular policy will eventually converge to a probability distribution over states $d(s)$
- $d(s)$ is called the stationary distribution over states of π
- $\sum_s d(s) = 1$
- $d(s)$ satisfies the following balance equation:

Markov

$$d(s') = \sum_s \sum_a \pi(a|s) p(s'|s,a) d(s)$$

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation

- Define the mean squared error of a linear value function approximation for a particular policy π relative to the true value given the distribution d as

$$MSVE_d(\mathbf{w}) = \sum_{s \in S} d(s)(V^\pi(s) - \hat{V}^\pi(s; \mathbf{w}))^2$$

- where
  - $d(s)$: stationary distribution of π in the true decision process
  - $\hat{V}^\pi(s; \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w}$, a linear value function approximation
- TD(0) policy evaluation with VFA converges to weights $\mathbf{w}_{TD}$ which is within a constant factor of the min mean squared error possible given distribution d:

$$MSVE_d(\mathbf{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\mathbf{w}} \sum_{s \in S} d(s)(V^\pi(s) - \hat{V}^\pi(s; \mathbf{w}))^2$$

Check Your Understanding L5N1: Poll

- TD(0) policy evaluation with VFA converges to weights $\boldsymbol{w}_{TD}$ which is within a constant factor of the min mean squared error possible for distribution d:

$$MSVE_d(\boldsymbol{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{w}} \sum_{s \in S} d(s)(V^\pi(s) - \hat{V}^\pi(s;\boldsymbol{w}))^2$$

- If the VFA is a tabular representation (one feature for each state), what is the $MSVE_d$ for TD?

1. Depends on the problem
2. MSVE = 0 for TD
3. Not sure

Check Your Understanding L5N1 : Poll

- TD(0) policy evaluation with VFA converges to weights $\mathbf{w}_{TD}$ which is within a constant factor of the min mean squared error possible for distribution d:

$$MSVE_d(\mathbf{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\mathbf{w}} \sum_{s \in S} d(s)(V^{\pi}(s) - \hat{V}^{\pi}(s;\mathbf{w}))^2$$

- If the VFA is a tabular representation (one feature for each state), what is the $MSVE_d$ for TD?

MSVE = 0 for TD

s.t. std cond on α

Table of Contents

Recall Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- In Monte Carlo methods, use a return G_t as a substitute target

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$$

- For SARSA instead use a TD target $r + \gamma \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the current function approximation value

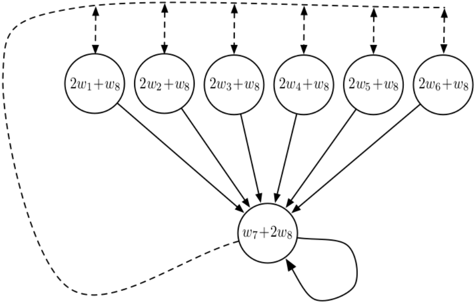
$$\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

- For Q-learning instead use a TD target $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the max of the current function approximation value

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

Challenges of Off Policy Control: Baird Example [1]

~1994/1995



$\pi(\text{solid}|\cdot) = 1$

$\mu(\text{dashed}|\cdot) = 6/7$

$\mu(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$

- Behavior policy and target policy are not identical
- Value can diverge

bootstrapping
func approx
off policy

Convergence of TD Methods with VFA

- Informally, updates involve doing an (approximate) Bellman backup followed by best trying to fit underlying value function to a particular feature representation
- Bellman operators are contractions, but value function approximation fitting can be an expansion
- Geoff Gordon 1995.

$$\|BV_1 - BV_2\|_\infty \le \|V_1 - V_2\|_\infty$$

$$\|O BV_1 - O BV_2\| \not\le \|V_1 - V_2\|_\infty$$

↑ func proj

Convergence of Policy Evaluation and Control Methods with VFA

convergence

Algorithm	Tabular	Linear VFA	General VFA
Monte-Carlo Control	✓	MC exploring starts 2022	✗
Sarsa	✓	chatter	✗
Q-learning	✓	✗	✗

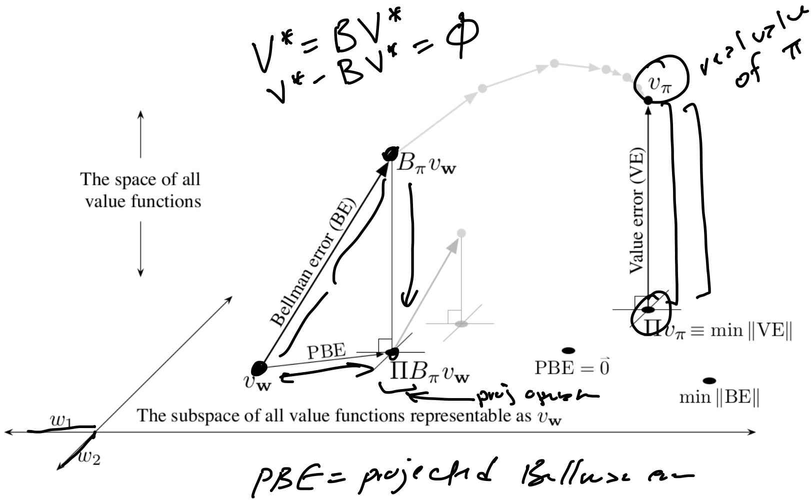
ϵ

α

Active Area: Off Policy Learning with Function Approximation

- Extensive work in better TD-style algorithms with value function approximation, some with convergence guarantees: see Chp 11 SB
- Will come up further later in this course

Value Function Approximation[1]



[1] Figure from Sutton and Barto 2018

Table of Contents

Table of Contents

Maximization Bias[2]

episodes are 1 step

- Consider single-state MDP ($|S| = 1$) with 2 actions, and both actions have 0-mean random rewards, ($\mathbb{E}(r|a = a_1) = \mathbb{E}(r|a = a_2) = 0$).
- Then $Q(s, a_1) = Q(s, a_2) = 0 = V(s)$
- Assume there are prior samples of taking action a_1 and a_2
- Let $\hat{Q}(s, a_1), \hat{Q}(s, a_2)$ be the finite sample estimate of Q
- Use an unbiased estimator for Q: e.g. $\hat{Q}(s, a_1) = \frac{1}{n(s,a_1)} \sum_{i=1}^{n(s,a_1)} r_i(s, a_1)$
- Let $\hat{\pi} = \arg\max_a \hat{Q}(s, a)$ be the greedy policy w.r.t. the estimated $\hat{Q}$

a_1: -.1 .5 .1 ...
a_2: -.5 .3 .2 .05 ...

$$\hat{V}^{\hat{\pi}} = E[\max \hat{Q}(s,a_1), \hat{Q}(s,a_2)]$$
$$\geq \max E[\hat{Q}(s,a_1)\ \hat{Q}(s,a_2)]$$
$$= \max(0, 0)$$
$$= 0 = V^{\pi}$$

Jensen's inequality

[2]Example from Mannor, Simester, Sun and Tsitsiklis. Bias and Variance Approximation in Value Function Estimates. Management Science 2007

Maximization Bias[3] Proof

- Consider single-state MDP ($|S| = 1$) with 2 actions, and both actions have 0-mean random rewards, ($\mathbb{E}(r|a = a_1) = \mathbb{E}(r|a = a_2) = 0$).
- Then $Q(s, a_1) = Q(s, a_2) = 0 = V(s)$
- Assume there are prior samples of taking action a_1 and a_2
- Let $\hat{Q}(s, a_1), \hat{Q}(s, a_2)$ be the finite sample estimate of Q
- Use an unbiased estimator for Q: e.g. $\hat{Q}(s, a_1) = \frac{1}{n(s,a_1)} \sum_{i=1}^{n(s,a_1)} r_i(s, a_1)$
- Let $\hat{\pi} = \arg\max_a \hat{Q}(s, a)$ be the greedy policy w.r.t. the estimated $\hat{Q}$
- *Even though each estimate of the state-action values is unbiased*, the estimate of $\hat{\pi}$'s value $\hat{V}^{\hat{\pi}}$ can be biased:
  $\hat{V}^{\hat{\pi}}(s) = \mathbb{E}[\max \hat{Q}(s, a_1), \hat{Q}(s, a_2)]$
  $\geq \max[\mathbb{E}[\hat{Q}(s, a_1)], [\hat{Q}(s, a_2)]]$
  $= max[0, 0] = V^{\pi}$,
  where the inequality comes from Jensen's inequality.

[3] Example from Mannor, Simester, Sun and Tsitsiklis. Bias and Variance Approximation in Value Function Estimates. Management Science 2007

Table of Contents

Double Q-Learning

- The greedy policy w.r.t. estimated Q values can yield a maximization bias during finite-sample learning
- Avoid using max of estimates as estimate of max of true values
- Instead split samples and use to create two independent unbiased estimates of $Q_1(s_1, a_i)$ and $Q_2(s_1, a_i)$ $\forall a$.
  - Use one estimate to select max action: $a^* = \arg\max_a Q_1(s_1, a)$
  - Use other estimate to estimate value of a^*: $Q_2(s, a^*)$
  - Yields unbiased estimate: $\mathbb{E}(Q_2(s, a^*)) = Q(s, a^*)$

Double Q-Learning

- The greedy policy w.r.t. estimated Q values can yield a maximization bias during finite-sample learning
- Avoid using max of estimates as estimate of max of true values
- Instead split samples and use to create two independent unbiased estimates of $Q_1(s_1, a_i)$ and $Q_2(s_1, a_i)$ $\forall a$.
  - Use one estimate to select max action: $a^* = \arg\max_a Q_1(s_1, a)$
  - Use other estimate to estimate value of a^*: $Q_2(s, a^*)$
  - Yields unbiased estimate: $\mathbb{E}(Q_2(s, a^*)) = Q(s, a^*)$
- Why is this an unbiased estimate of the max state-action value?

Double Q-Learning

- The greedy policy w.r.t. estimated Q values can yield a maximization bias during finite-sample learning
- Avoid using max of estimates as estimate of max of true values
- Instead split samples and use to create two independent unbiased estimates of $Q_1(s_1, a_i)$ and $Q_2(s_1, a_i)$ $\forall a$.
  - Use one estimate to select max action: $a^* = \arg\max_a Q_1(s_1, a)$
  - Use other estimate to estimate value of a^*: $Q_2(s, a^*)$
  - Yields unbiased estimate: $\mathbb{E}(Q_2(s, a^*)) = Q(s, a^*)$
- Why is this an unbiased estimate of the max state-action value? Using independent samples to estimate the value
- If acting online, can alternate samples used to update Q_1 and Q_2, using the other to select the action chosen
- Next slides extend to full MDP case (with more than 1 state)

Double Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
loop
    Select a_t using ϵ-greedy $\pi(s) = \arg\max_a Q_1(s_t, a) + Q_2(s_t, a)$
    Observe (r_t, s_{t+1})
    if (with 0.5 probability) **then**
        $Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha(r_t + \gamma Q_2(s_{t+1}, \arg\max_a Q_1(s_{t+1}, a)) - Q_1(s_t, a_t))$
    else
        $Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha(r_t + \gamma Q_1(s_{t+1}, \arg\max_a Q_2(s_{t+1}, a)) - Q_2(s_t, a_t))$
    end if
    $t = t + 1$
end loop

Compared to Q-learning, how does this change the: memory requirements, 2x
computation requirements per step, amount of data required?

doesn't change

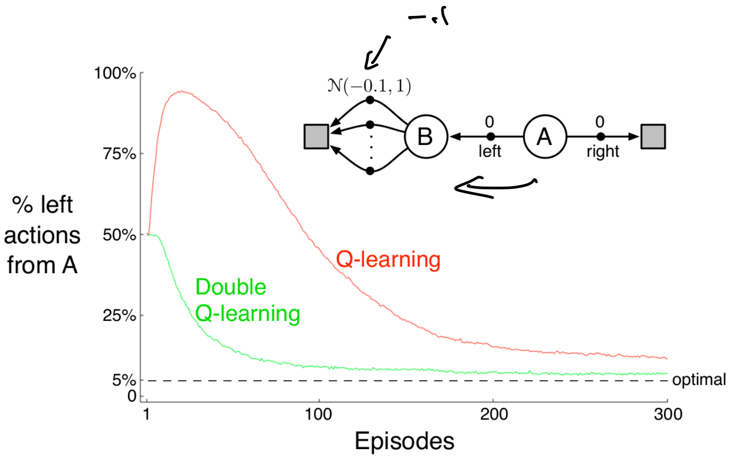
Double Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a)$,$\forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
loop
    Select a_t using ϵ-greedy $\pi(s) = \arg\max_a Q_1(s_t, a) + Q_2(s_t, a)$
    Observe (r_t, s_{t+1})
    if (with 0.5 probability) **then**
        $Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha(r_t + \gamma Q_2(s_{t+1}, \arg\max_a Q_1(s_{t+1}, a)) - Q_1(s_t, a_t))$
    else
        $Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha(r_t + \gamma Q_1(s_{t+1}, \arg\max_a Q_2(s_{t+1}, a)) - Q_2(s_t, a_t))$
    end if
    $t = t + 1$
end loop

Compared to Q-learning, how does this change the: memory requirements, computation requirements per step, amount of data required?

Doubles the memory, same computation requirements, data requirements are subtle– might reduce amount of exploration needed due to lower bias

Double Q-Learning (Figure 6.7 in Sutton and Barto 2018)



Due to the maximization bias, Q-learning spends much more time selecting suboptimal actions than double Q-learning.

Table of Contents

Recall DQN

- Deep Q-learning (DQN): Q-learning with deep neural networks **and**
  - Experience replay
  - Fixed Q-targets

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^{-}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

Recall DQN Pseudocode

Input $\boldsymbol{C}$, $\boldsymbol{\alpha}$, $D = \{\}$, Initialize $\boldsymbol{w}$, $\boldsymbol{w}^- = \boldsymbol{w}$, $t = 0$
Get initial state s_0
loop
    Sample action a_t given ϵ-greedy policy for current $\hat{Q}(s_t, a; \boldsymbol{w})$
    Observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer D
    Sample random minibatch of tuples (s_i, a_i, r_i, s_{i+1}) from D
    for j in minibatch **do**
        if episode terminated at step $i + 1$ **then**
            $y_i = r_i$
        else
            $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \boldsymbol{w}^-)$
        end if
        Do gradient descent step on $(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))^2$ for parameters $\boldsymbol{w}$: $\Delta \boldsymbol{w} = \alpha(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_i, a_i; \boldsymbol{w})$
    end for
    $t = t + 1$
    if mod(t,C) == 0 **then**
        $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$
    end if
end loop

Double DQN

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Extend double Q learning to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^-$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}; \boldsymbol{w}^-)}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{Q}(s, a; \boldsymbol{w}))$$

Double DQN

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Extend double Q learning to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^{-}$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}; \boldsymbol{w}^{-})}^{\text{Action evaluation: } \boldsymbol{w}^{-}} - \hat{Q}(s, a; \boldsymbol{w}))$$

- How is this different from fixed target network update used in DQN?

Double DQN

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Extend double Q learning to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^-$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}; \boldsymbol{w}^-)}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{Q}(s, a; \boldsymbol{w}))$$

- How is this different from fixed target network update used in DQN? In DQN the same weights $\boldsymbol{w}^-$ were used to choose the best action at s' and evaluate its value $\hat{Q}(s', a'; \boldsymbol{w}^-)$

Double DQN

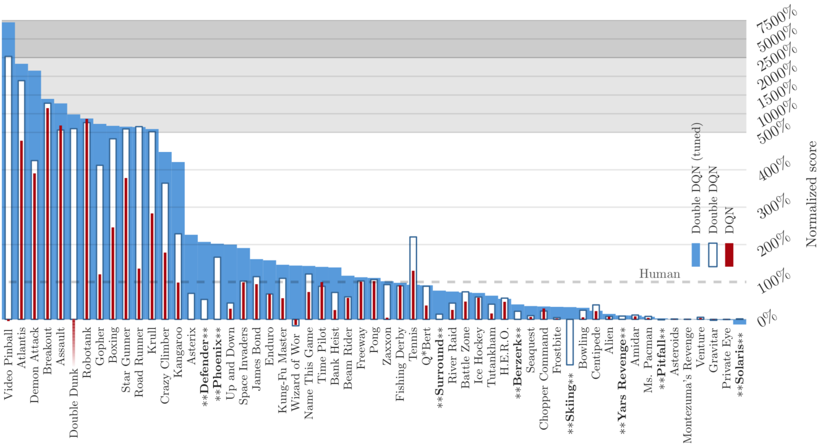


Figure: van Hasselt, Guez, Silver, 2015

Double DQN

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Extend double Q learning to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^-$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}; \boldsymbol{w}^-)}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{Q}(s, a; \boldsymbol{w}))$$

- **Very small code change, often can lead to significantly improved results**

Table of Contents

Rainbow: Combining Improvements in Deep Reinforcement Learning. Hessel et al. 2018 (DeepMind)

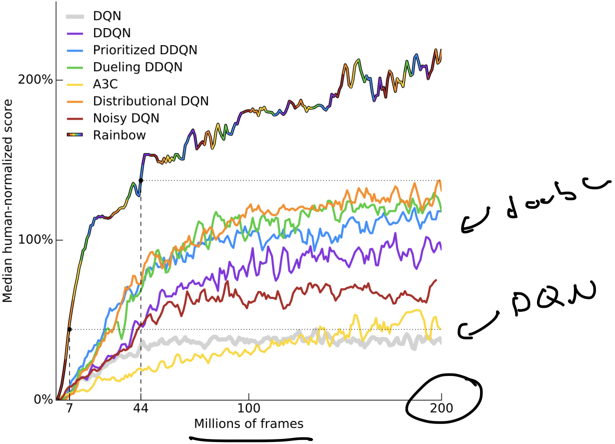


Figure: Median human-normalized performance across 57 Atari games. Curves smoothed with a moving avg over 5 points.

Many new methods

- One (of many) significant ideas: use additional objectives

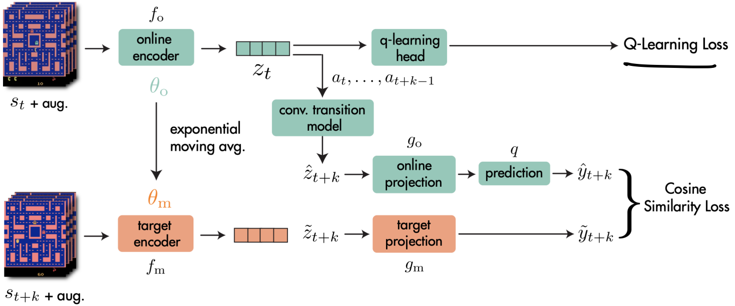


Figure: Data-efficient reinforcement learning with self-predictive representations. Schwarzer et al. ICLR 2021.

What is Enabling Progress?

- Benchmark tasks. Atari, Atari 100k, Mujoco, ...
- Standing on the shoulders of giants... : building on past algorithms
  - and code bases for said algorithms

Model-free value function approximation RL: What You Should Know

- Be able to derive weight update for generic function approximation for Q/V^π
- Understand various (MC/SARSA/Q-learning) targets used when updating Q function
- Know what TD vs MC converge to for policy evaluation with a linear function approximator
- Be able to implement DQN
- Define the maximization bias and give one tool for alleviating it

Class Structure

- Last time: Model-free value function approximation control and Deep Q-learning
- This time: Model-free value function approximation and more DQN
- Next time: Policy search in large spaces / policy gradient methods

Lecture 6: Refresh Your Knowledge

- In TD learning with linear VFA (select all):
  1. $\boldsymbol{w} = \boldsymbol{w} + \alpha(r(s_t) + \gamma \boldsymbol{x}(s_{t+1})^T \boldsymbol{w} - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t)$
  2. $V(s) = \boldsymbol{w}(s)\boldsymbol{x}(s)$
  3. Asymptotic convergence to the true best minimum MSE linear representable $V(s)$ is guaranteed for $\alpha \in (0, 1)$, $\gamma < 1$.
  4. Not sure

Lecture 6: Refresh Your Knowledge **Solutions**

- In TD learning with linear VFA (select all):
  1. $\mathbf{w} = \mathbf{w} + \alpha(r(s_t) + \gamma \mathbf{x}(s_{t+1})^T \mathbf{w} - \mathbf{x}(s_t)^T \mathbf{w})\mathbf{x}(s_t)$
  2. $V(s) = \mathbf{w}(s)\mathbf{x}(s)$
  3. Asymptotic convergence to the true best minimum MSE linear representable $V(s)$ is guaranteed for $\alpha \in (0, 1)$, $\gamma < 1$.
  4. Not sure

Answer: 1 is true. Convergence is not guaranteed to the best, the resulting one may still be worse than the best MSE solution by a factor of $\frac{1}{1-\gamma}$. It is also important to know that this is with respect to the stationary distirbution $d(s)$. Also note the weights do not depend on the state.