

DEEP Q-LEARNING

DQN Enhancements

Diego Klabjan
Professor, Industrial Engineering and Management Sciences

Northwestern | McCORMICK SCHOOL OF
ENGINEERING

REVIEW

Functional Approximation of V

- Updating the value function approximation
- At state s
 - Have reward plus future based on approximate value function
 - Based on optimal action
 - Have value of approximate value function
- Match them
 - L2 loss

fitted value iteration algorithm:

1. set $y_i \leftarrow \max_{a_i} (r(s_i, a_i) + \gamma E[V_\phi(s'_i)])$
2. set $\phi \leftarrow \operatorname{argmin}_\phi \frac{1}{2} \sum_i \|V_{\phi'}(s_i) - y_i\|^2$

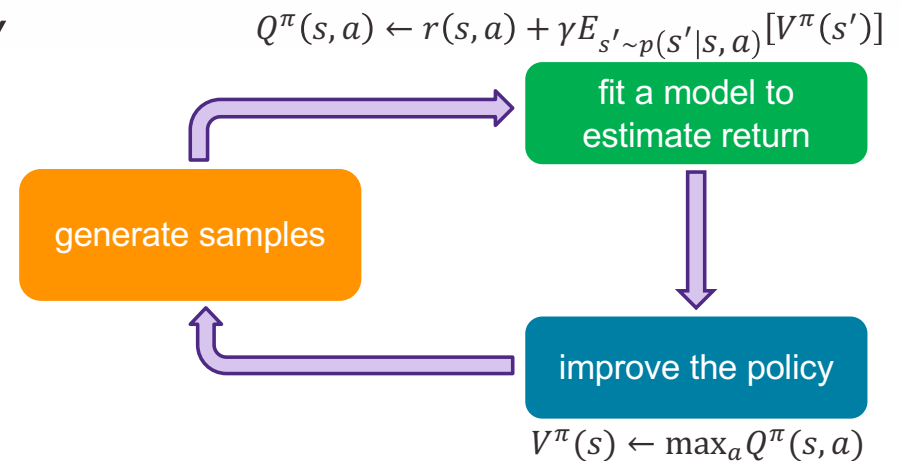
Putting it all Together

- For $k = 1, 2, \dots$
 - For $n = 0, 1, 2, \dots, N$
 - Sample s_n
 - For $n = 0, 1, 2, \dots, N$
 - Compute $y_n = \max_a (r(s_n, a) + \gamma E[V_\phi(s'_n | s_n)])$
 - Set $\phi \leftarrow \operatorname{argmin}_\phi \frac{1}{2} \sum_n \|V_{\phi'}(s_n) - y_n\|^2$
- Last step
 - Standard gradient optimization by back propagation
- Drawback
 - No policy improvement

$$\begin{array}{c} s_n \\ \hline r(s_n, a) + \gamma E[V_\phi(s'_n | s_n)] \\ \hline V_{\phi'}(s_n) \end{array}$$

Revised

- Value function as neural network
- Keep track of Q-factor at samples generated
- Compute $Q^\pi(s, a)$ at generated samples
 - Samples generated based on incumbent policy
 - Use relationship between Q and V
- Update policy in greedy manner from Q



Algorithm

- For $k = 1, 2, \dots$
 - For $n = 0, 1, 2, \dots, N$
 - Sample $s_n, a_n \sim \pi(a_n | s_n)$
 - For $n = 0, 1, 2, \dots, N$
 - $Q^\pi(s_n, a_n) \leftarrow r(s_n, a_n) + \gamma E_{s' \sim p(s' | s_n, a_n)} [V_\phi(s')]$
 - Compute $y_n = \max_a (r(s_n, a) + \gamma E_{s' \sim p(s' | s_n, a)} [V_\phi(s')])$
 - Set $\phi \leftarrow \operatorname{argmin}_\phi \frac{1}{2} \sum_n \|V_{\phi'}(s_n) - y_n\|^2$
 - $\pi(s) \leftarrow \operatorname{argmax}_a Q^\pi(s, a)$

Unknown Transition Function

- Observe only episodes – imitation learning
- Functional approximation of transition function not possible
 - Output state

fitted value iteration algorithm:

1. set $y_i \leftarrow \max_{a_i} (r(s_i, a_i) + \gamma E[V_\phi(s'_i)])$
 2. set $\phi \leftarrow \operatorname{argmin}_\phi \frac{1}{2} \sum_i \|V_{\phi'}(s_i) - y_i\|^2$
- Need to know outcomes for different actions!

policy iteration algorithm

1. evaluate $Q^\pi(s, a)$
2. set $\pi \leftarrow \pi'$

$$\pi'(a_t | s_t) = \begin{cases} 1 & \text{if } a_t = \operatorname{argmax}_{a_t} Q^\pi(s_t, a_t) \\ 0 & \text{otherwise} \end{cases}$$

policy evaluation:

$$V^\pi(s) \leftarrow r(s, \pi(s)) + \gamma E_{s' \sim p(s' | s, \pi(s))} [V^\pi(s')]$$

$$Q^\pi(s, a) \leftarrow r(s, a) + \gamma E_{s' \sim p(s' | s, \pi(s))} [Q^\pi(s', \pi(s'))]$$

Functional Approximation of Q

policy iteration:

1. evaluate $V^\pi(s)$
2. set $\pi \leftarrow \pi'$

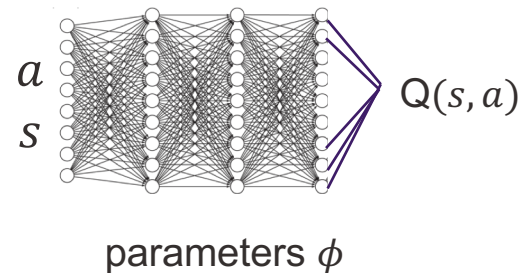
fitted value iteration algorithm:

1. set $y_i \leftarrow \max_{a_i}(r(s_i, a_i) + \gamma E[V_\phi(s'_i)])$
2. set $\phi \leftarrow \operatorname{argmin}_\phi, \frac{1}{2} \sum_i \|V_{\phi'}(s_i) - y_i\|^2$

fitted Q iteration algorithm:

1. set $y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s'_i)]$ ← approximate $E[V(s'_i)] \approx \max_{a'} Q_\phi(s'_i, a')$
2. set $\phi \leftarrow \operatorname{argmin}_\phi, \frac{1}{2} \sum_i \|Q_{\phi'}(s_i, a_i) - y_i\|^2$

Doesn't require simulation of actions!



Value Iteration with Fitted Q-factor

- Observed data are trajectories
 - Formally, $U = \{(s_i, a_i, s'_i, r_i) | i \in N\}$
- Loop
 - Sample $S \subseteq U$
 - For $i \in S$
 - $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
 - Set $\phi \leftarrow \operatorname{argmin}_\phi \frac{1}{2} \sum_{i \in S} \|Q_\phi(s_i, a_i) - y_i\|^2$
- Only remaining drawback how to compute max

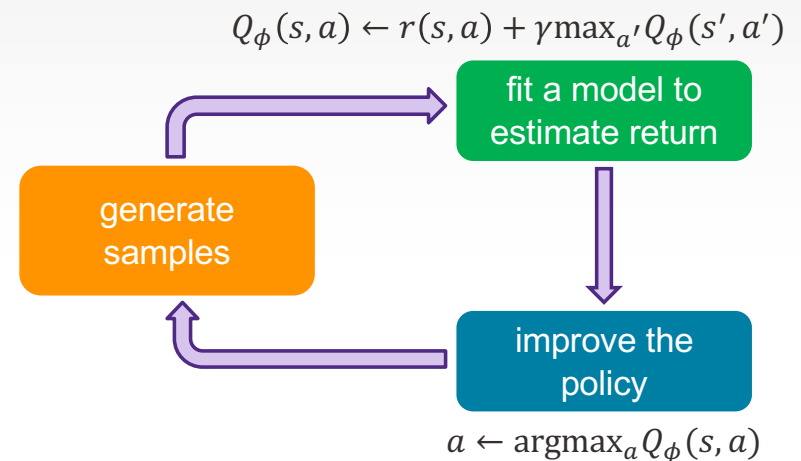
Online Version

full fitted Q-iteration algorithm:

1. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy
2. set $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a')$
3. set $\phi \leftarrow \operatorname{argmin}_\phi, \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

online Q iteration algorithm:

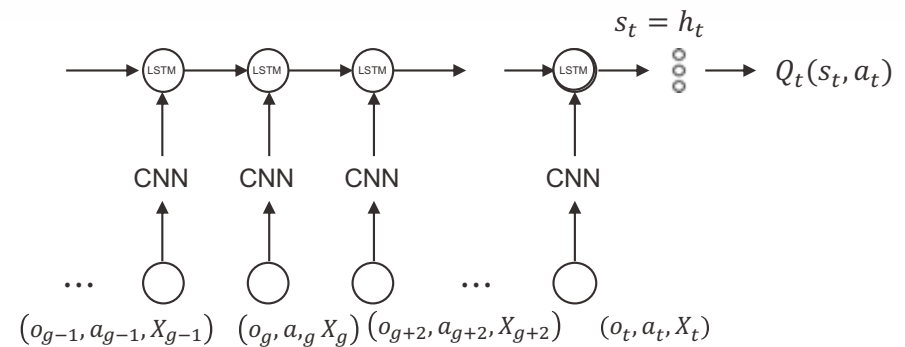
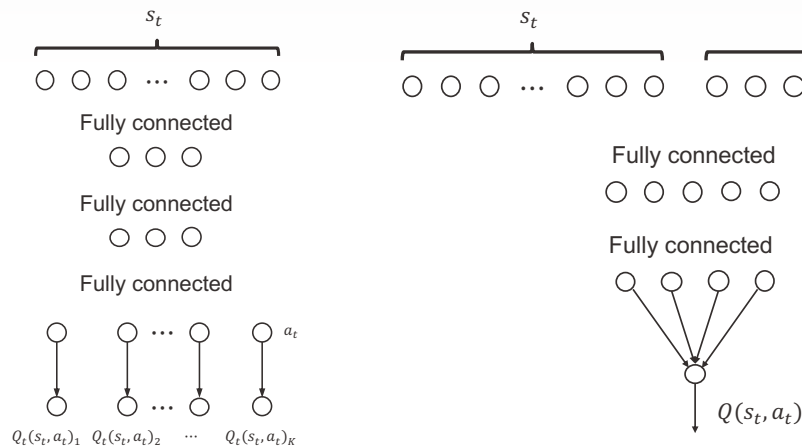
1. take some action a_i and observe (s_i, a_i, s'_i, r_i)
2. $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a')$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)$



DQN


Deep Q-Learning

- Deep network used to model Q factor
- State modeled by RNN
 - Possibly combined with CNN if images involved
- Final network a combination of CNN+RNN+Fully connected



Issues

Online Q iteration algorithm:

- 
1. take some action a_i and observe (s_i, a_i, s'_i, r_i)
 2. $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)$
- These are correlated!
- Gradient descent!
Convergence is a problem!
- Or it is not gradient descent!

Q-learning is not gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i) \right] \right)$$

No gradient through target value!

Correlated Samples in Online Q-learning

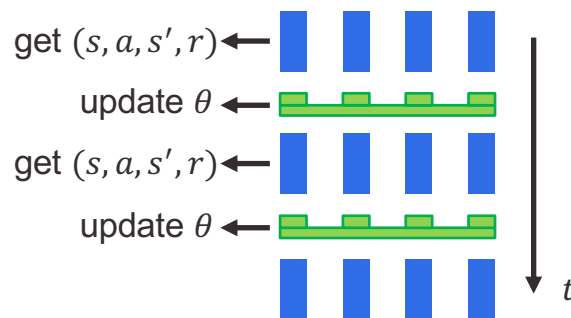
Online Q iteration algorithm:

- Sequential states are strongly correlated
- Target value is always changing

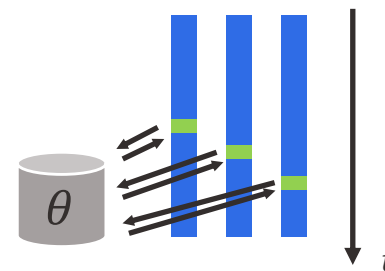
1. take some action a_i and observe (s_i, a_i, s'_i, r_i)
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i) \right] \right)$



Synchronized parallel



Asynchronous parallel



Replay Buffers

Online Q iteration algorithm:

Special case with $K=1$, and one gradient step

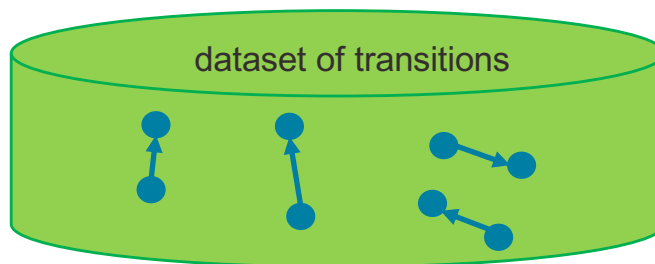
1. take some action a_i and observe (s_i, a_i, s'_i, r_i)
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i) \right] \right)$

Full fitted Q-iteration algorithm:

- ~~1. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy~~
2. set $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)$
- $K \times$ 3. set $\phi \leftarrow \operatorname{argmin}_\phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

Any policy will work! (with broad support)

Idea: for different policies store tuple in buffer
Just load data from a buffer here
Still use one gradient step



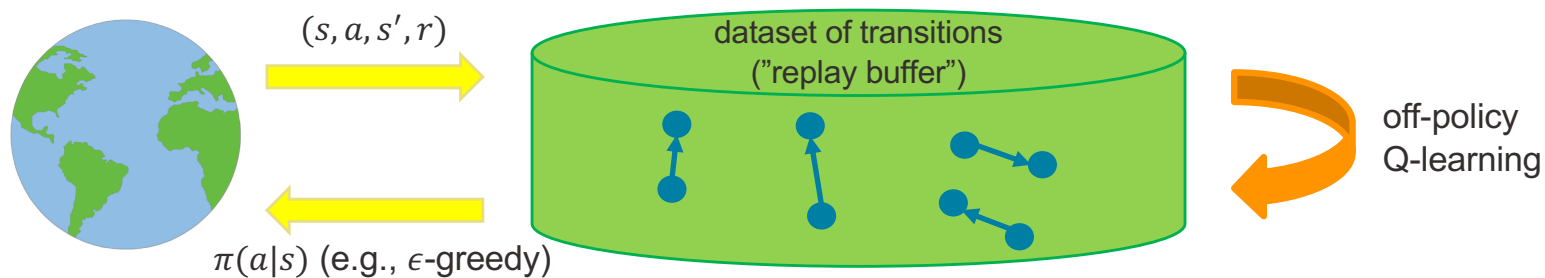
Fitted Q-iteration

Replay Buffers

Q learning with a replay buffer:

1. sample a batch (s_i, a_i, s'_i, r_i) from \mathcal{B} + samples are no longer correlated
2. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i) \right] \right)$
+ multiple samples in the batch (low-variance gradient)

Need to periodically feed the replay buffer

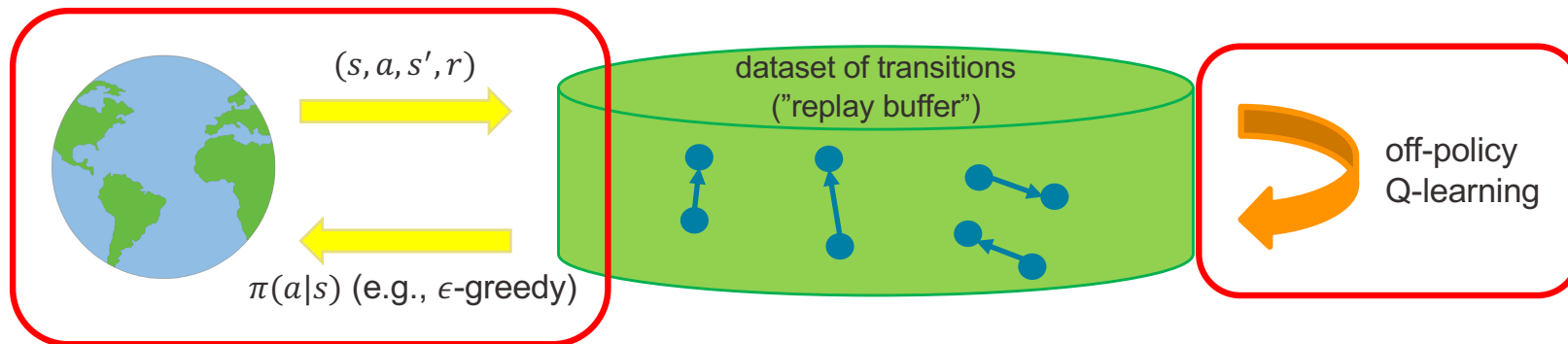


Framework

Full Q-learning with replay buffer:


1. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add it to \mathcal{B}
2. sample a batch (s_i, a_i, s'_i, r_i) from \mathcal{B}
3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i) \right] \right)$

K=1 is common



Issues Revisited

Online Q iteration algorithm:

- 
1. take some action a_i and observe (s_i, a_i, s'_i, r_i)
 2. $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a')$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)$
- ~~these are correlated!~~
Use replay buffer

Q-learning is not gradient descent!

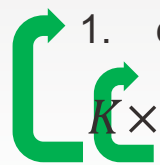
$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a') \right] \right)$$

This is still a problem!

no gradient through target value

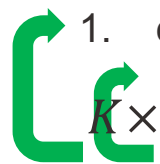
Q-Learning and Regression

Full Q-learning with replay buffer:

- 
1. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 2. sample a batch (s_i, a_i, s'_i, r_i) from \mathcal{B}
 3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i) \right] \right)$

One gradient step, moving target

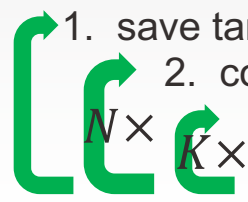
Full fitted Q-iteration algorithm:

- 
1. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy
 2. set $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
 3. set $\phi \leftarrow \operatorname{argmin}_\phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

Perfectly well-defined, stable regression

Q-Learning with Target Networks

Full Q-learning with replay buffer:

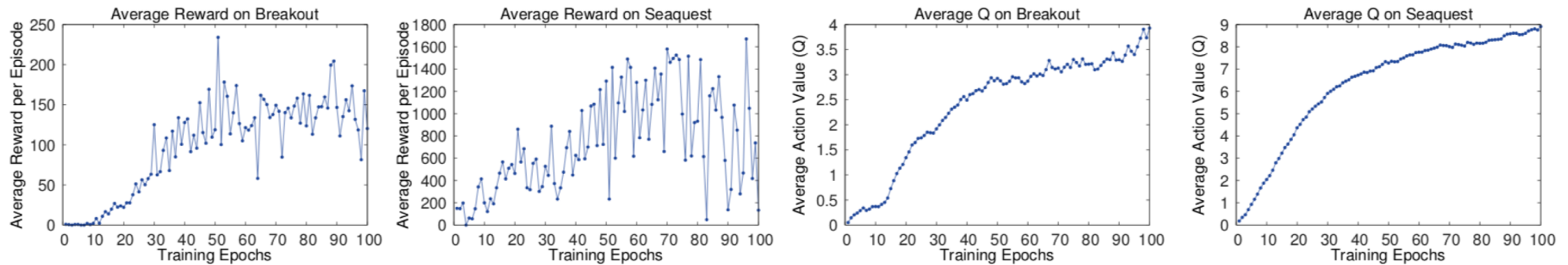
- 
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 - $N \times$ 3. sample a batch (s_i, a_i, s'_i, r_i) from \mathcal{B}
 - $K \times$ 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \underbrace{\left[r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a'_i) \right]}_{\text{Targets don't change in inner loop!}} \right)$
- supervised regression

Performance

	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

Mnih et al, Playing Atari with Deep Reinforcement Learning

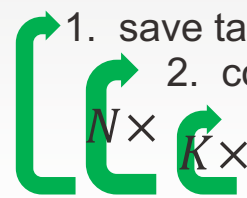
Performance



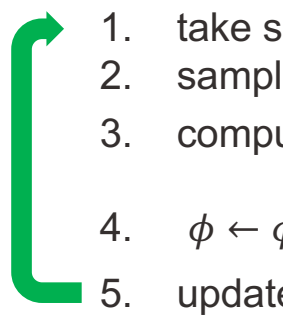
Mnih et al, Playing Atari with Deep Reinforcement Learning

“Classic” deep Q-learning Algorithm (DQN)

Q-learning with replay buffer and target network:

- 
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 - $N \times$ 3. sample a batch (s_i, a_i, s'_i, r_i) from \mathcal{B}
 - $K \times$ 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} (Q_{\phi'}(s'_i, a')) \right] \right)$

“Classic” deep Q-learning algorithm:

- 
1. take some action a_i and observe (s_i, a_i, s'_i, r_i) , add it to \mathcal{B}
 2. sample mini-batch $\{s_j, a_j, s'_j, r_j\}$ from \mathcal{B} uniformly
 3. compute $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$ using *target* network $Q_{\phi'}$
 4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_j, a_j) (Q_\phi(s_j, a_j) - y_j)$
 5. update ϕ' : copy ϕ every N steps
- } $K = 1$

Alternative Target Network

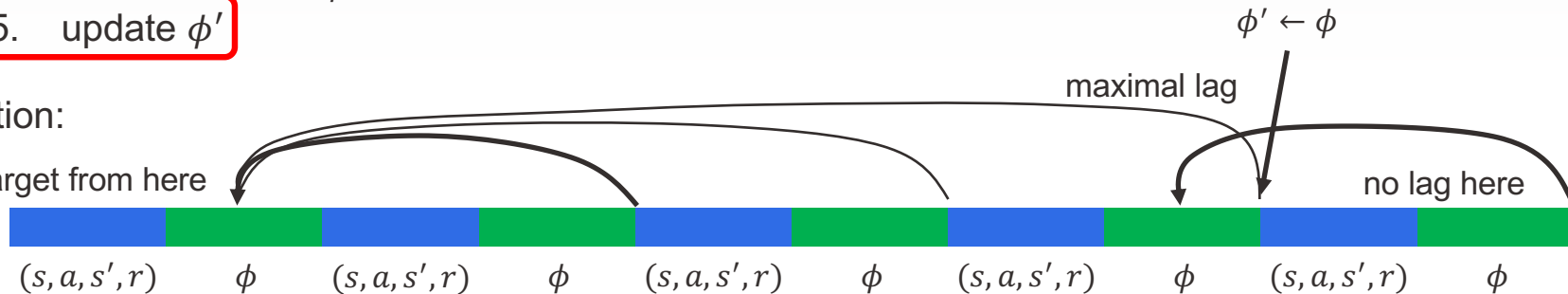
“Classic” deep Q-learning algorithm:

1. take some action a_i and observe (s_i, a_i, s'_i, r_i) , add it to \mathcal{B}
2. sample mini-batch $\{s_j, a_j, s'_j, r_j\}$ from \mathcal{B} uniformly
3. compute $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_j, a_j)(Q_\phi(s_j, a_j) - y_j)$
5. update ϕ'

$K = 1$

Intuition:

get target from here



Feels weirdly uneven, can we always have the same lag?

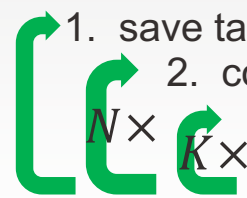
5. update ϕ' : $\phi' \leftarrow \tau\phi' + (1 - \tau)\phi$

$\tau = 0.999$ works well

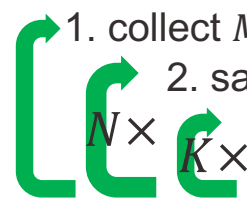
Fitted Q-iteration and Q-learning

Q-learning with replay buffer and target network:

DQN: $N = 1, K = 1$

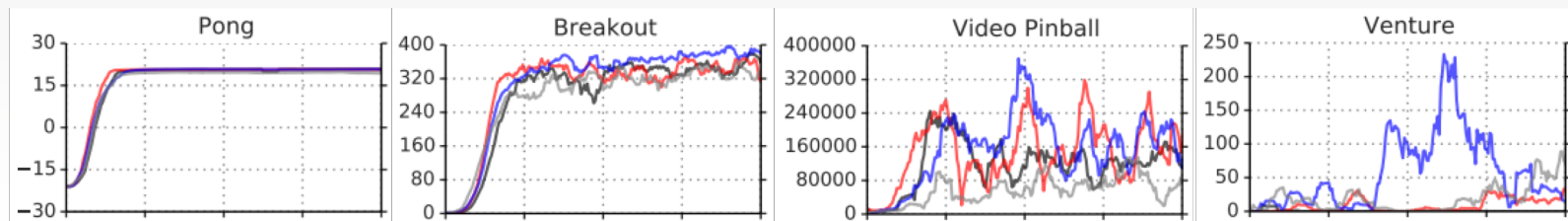
- 
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect M data points $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 - $N \times$ 3. sample a batch (s_i, a_i, s'_i, r_i) from \mathcal{B}
 - $K \times$ 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a'_i) \right] \right)$

Fitted Q-learning (written similarly as above):

- 
1. collect M data points $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 2. save target network parameters: $\phi' \leftarrow \phi$
 - $N \times$ 3. sample a batch (s_i, a_i, s'_i, r_i) from \mathcal{B}
 - $K \times$ 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a'_i) \right] \right)$
- } just SGD

Simple Practical Tips for Q-learning

- Q-learning takes some care to stabilize
 - Test on easy, reliable tasks first, make sure your implementation is correct

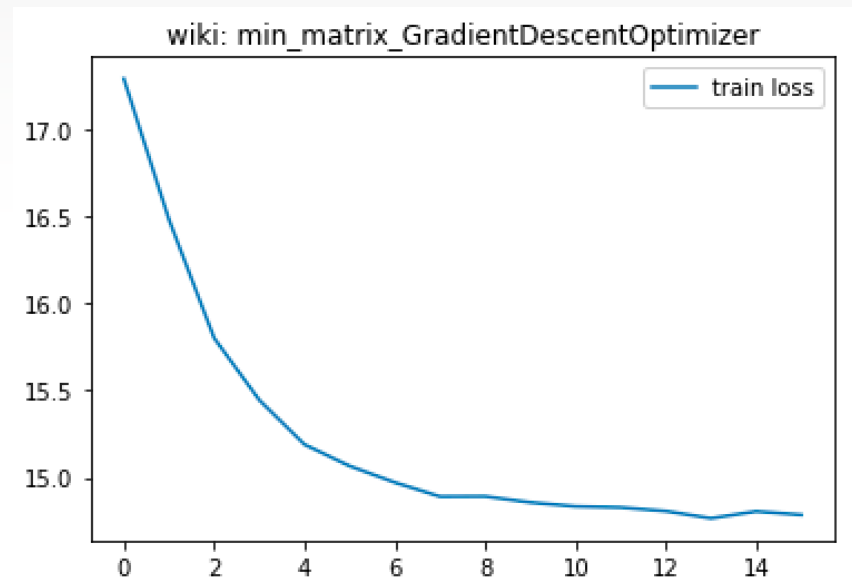


T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". [arXiv:1511.05952](https://arxiv.org/abs/1511.05952), Figure 7

- Large replay buffers help improve stability
 - Looks more like fitted Q-iteration
- It takes time, be patient—might be no better than random for a while
- Start with high exploration (epsilon) and gradually reduce

Simple Practical Tips for Q-learning

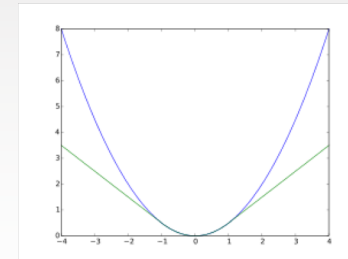
- Standard SGD in supervised learning
 - Loss drop drastic at the beginning
- DQN
 - Slow at the beginning
 - S-curve



Advanced Tips for Q-learning

- Bellman error gradients can be big; clip gradients or use Huber loss

$$L(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \leq \delta \\ \delta|x| - \frac{\delta^2}{2} & \text{otherwise} \end{cases}$$



- Double Q-learning helps a lot in practice
 - Did not cover
- N-step returns also help a lot
 - Have issues
 - Did not cover
- Schedule exploration (high to low) and learning rates (high to low)
 - Adam optimizer can help too
- Run multiple random seeds
 - Very inconsistent between runs