Today: Model-free Control

- Generalized policy improvement
- Importance of exploration
- Monte Carlo control
- Model-free control with temporal difference (SARSA, Q-learning)
- Maximization bias

Model-free Control Examples

- Many applications can be modeled as a MDP: Backgammon, Go, Robot locomation, Helicopter flight, Robocup soccer, Autonomous driving, Customer ad selection, Invasive species management, Patient treatment
- For many of these and other problems either:
  - MDP model is unknown but can be sampled
  - MDP model is known but it is computationally infeasible to use directly, except through sampling

On and Off-Policy Learning

- On-policy learning
  - Direct experience
  - Learn to estimate and evaluate a policy from experience obtained from following that policy
- Off-policy learning
  - Learn to estimate and evaluate a policy using experience gathered from following a different policy

Table of Contents

Recall Policy Iteration

- Initialize policy π
- Repeat:
  - Policy evaluation: compute V^{π}
  - Policy improvement: update π

$$\pi'(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s') = \arg\max_a Q^{\pi}(s,a) \tag{1}$$

- Now want to do the above two steps without access to the true dynamics and reward models
- Last lecture introduced methods for model-free policy evaluation

Model-free Generalized Policy Improvement

- Given an estimate $Q^{\pi_i}(s, a)\ \forall s, a$
- Update new policy

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s, a) \tag{2}$$

Model-free Policy Iteration

- Initialize policy π
- Repeat:
  - Policy evaluation: compute Q^{π}
  - Policy improvement: update π given Q^{π}

- May need to modify policy evaluation:
  - If π is deterministic, can't compute $Q(s, a)$ for any $a \neq \pi(s)$
- How to interleave policy evaluation and improvement?
  - Policy improvement is now using an estimated Q

Table of Contents

Policy Evaluation with Exploration

- Want to compute a model-free estimate of Q^{π}
- In general seems subtle
  - Need to try all (s, a) pairs but then follow π
  - Want to ensure resulting estimate Q^{π} is good enough so that policy improvement is a monotonic operator
- For certain classes of policies can ensure all (s,a) pairs are tried such that asymptotically Q^{π} converges to the true value

ϵ-greedy Policies

- Simple idea to balance exploration and exploitation
- Let $|A|$ be the number of actions
- Then an ϵ-greedy policy w.r.t. a state-action value $Q^{\pi}(s, a)$ is $\pi(a|s) =$

Monotonic[19] ϵ-greedy Policy Improvement

Theorem

For any ϵ-greedy policy π_i, the ϵ-greedy policy w.r.t. Q^{π_i}, π_{i+1} is a monotonic improvement $V^{\pi_{i+1}} \geq V^{\pi}$

$$
\begin{aligned}
Q^{\pi}(s, \pi_{i+1}(s)) &= \sum_{a \in A} \pi_{i+1}(a|s) Q^{\pi_i}(s, a) \\
&= (\epsilon/|A|) \sum_{a \in A} Q^{\pi_i}(s, a) + (1 - \epsilon) \max_a Q^{\pi_i}(s, a)
\end{aligned}
$$

- Therefore $V^{\pi_{i+1}} \geq V^{pi}$ (from the policy improvement theorem)

[19] The theorem assumes that Q^{π_i} has been computed exactly.

Monotonic[21] ϵ-greedy Policy Improvement

Theorem

For any ϵ-greedy policy π_i, the ϵ-greedy policy w.r.t. Q^{π_i}, π_{i+1} is a monotonic improvement $V^{\pi_{i+1}} \geq V^{\pi}$

$$
\begin{aligned}
Q^{\pi}(s, \pi_{i+1}(s)) &= \sum_{a \in A} \pi_{i+1}(a|s) Q^{\pi_i}(s, a) \\
&= (\epsilon/|A|) \sum_{a \in A} Q^{\pi_i}(s, a) + (1 - \epsilon) \max_a Q^{\pi_i}(s, a) \\
&= (\epsilon/|A|) \sum_{a \in A} Q^{\pi_i}(s, a) + (1 - \epsilon) \max_a Q^{\pi_i}(s, a) \frac{1 - \epsilon}{1 - \epsilon} \\
&= (\epsilon/|A|) \sum_{a} Q^{\pi_i}(s, a) + (1 - \epsilon) \max_a Q^{\pi_i}(s, a) \sum_a \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1 - \epsilon} \\
&\geq \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s, a) + (1 - \epsilon) \sum_a \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1 - \epsilon} Q^{\pi_i}(s, a) \\
&= \sum_a \pi_i(a|s) Q^{\pi_i}(s, a) = V^{\pi_i}(s)
\end{aligned}
$$

- Therefore $V^{\pi_{i+1}} \geq V^{pi}$ (from the policy improvement theorem)

[21] The theorem assumes that Q^{π_i} has been computed exactly.

Greedy in the Limit of Infinite Exploration (GLIE)

Definition of GLIE

- All state-action pairs are visited an infinite number of times

$$\lim_{i \to \infty} N_i(s, a) \to \infty$$

- Behavior policy converges to greedy policy

- A simple GLIE strategy is ϵ-greedy where ϵ is reduced to 0 with the following rate: $\epsilon_i = 1/i$

Table of Contents

Monte Carlo Online Control / On Policy Improvement

1: Initialize $Q(s,a) = 0, Returns(s,a) = 0 \; \forall (s,a)$, Set $\epsilon = 1$, $k = 1$
2: $\pi_k = \epsilon$-greedy(Q) // Create initial ϵ-greedy policy
3: **loop**
4: Sample k-th episode $(s_{k1}, a_{k1}, r_{k1}, s_{k2}, \ldots, s_T)$ given π_k
5: **for** $t = 1, \ldots, T$ **do**
6: **if** First visit to (s,a) in episode k **then**
7: Append $\sum_{j=t}^{T} r_{kj}$ to $Returns(s_t, a_t)$
8: $Q(s_t, a_t) = average(Returns(s_t, a_t))$
9: **end if**
10: **end for**
11: $k = k + 1$, $\epsilon = 1/k$
12: $\pi_k = \epsilon$-greedy(Q^{π}) // Policy improvement
13: **end loop**

GLIE Monte-Carlo Control

Theorem

GLIE Monte-Carlo control converges to the optimal state-action value[a] function $Q(s,a) \rightarrow q(s,a)$

---

[a] $v(s)$ and $q(s,a)$ without any additional subscripts are used to indicate the optimal state and state-action value function, respectively.

Model-free Policy Iteration

- Initialize policy π
- Repeat:
  - Policy evaluation: compute Q^{π}
  - Policy improvement: update π given Q^{π}

- What about TD methods?

Table of Contents

Model-free Policy Iteration with TD Methods

- Use temporal difference methods for policy evaluation step
- Initialize policy π
- Repeat:
  - Policy evaluation: compute Q^{π} using temporal difference updating with ϵ-greedy policy
  - Policy improvement: Same as Monte carlo policy improvement, set π to ϵ-greedy (Q^{π})

General Form of SARSA Algorithm

1: Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$
2: Take $a_t \sim \pi(s_t)$ // Sample action from policy
3: Observe (r_t, s_{t+1})
4: **loop**
5: Take action $a_{t+1} \sim \pi(s_{t+1})$
6: Observe (r_{t+1}, s_{t+2})
7: Update Q given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$:
8: Perform policy improvement:
9: $t = t + 1$
10: **end loop**

- What are the benefits to improving the policy after each step?
- What are the benefits to updating the policy less frequently?

Convergence Properties of SARSA

Theorem

Sarsa for finite-state and finite-action MDPs converges to the optimal action-value, $Q(s,a) \rightarrow q(s,a)$, under the following conditions:

1. The policy sequence $\pi_t(a|s)$ satisfies the condition of GLIE
2. The step-sizes α_t satisfy the Robbins-Munro sequence such that

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Recall: Off Policy, Policy Evaluation

- Given data from following a behavior policy π_b can we estimate the value V^{π_e} of an alternate policy π_b?
- Neat idea: can we learn about other ways to do things different than what we actually did?
- Discussed how to do this for Monte Carlo evaluation
- Used Importance Sampling
- First see how to do off policy evaluation with TD

Importance Sampling for Off Policy TD (Policy Evaluation)

- Recall the Temporal Difference (TD) algorithm which is used to incremental model-free evaluation of a policy π_b. Precisely, given a state s_t, an action a_t sampled from $\pi_b(s_t)$ and the observed reward r_t and next state s_{t+1}, TD performs the following update:

$$V^{\pi_b}(s_t) = V^{\pi_b}(s_t) + \alpha(r_t + \gamma V^{\pi_b}(s_{t+1}) - V^{\pi_b}(s_t)) \tag{3}$$

- Now want to use data generated from following π_b to estimate the value of different policy π_e, V^{π_e}
- Change TD target $r_t + \gamma V(s_{t+1})$ to weight target by single importance sample ratio
- New update:

$$V^{\pi_e}(s_t) = V^{\pi_e}(s_t) + \alpha \left[\frac{\pi_e(a_t|s_t)}{\pi_b(a_t|s_t)} (r_t + \gamma V^{\pi_e}(s_{t+1}) - V^{\pi_e}(s_t)) \right] \tag{4}$$

Importance Sampling for Off Policy TD Cont.

- Off Policy TD Update:

$$V^{\pi_e}(s_t) = V^{\pi_e}(s_t) + \alpha \left[\frac{\pi_e(a_t|s_t)}{\pi_b(a_t|s_t)} (r_t + \gamma V^{\pi_e}(s_{t+1}) - V^{\pi_e}(s_t)) \right] \quad (5)$$

- Significantly lower variance than MC IS. (Why?)

- Does π_b need to be the same at each time step?

- What conditions on π_b and π_e are needed for off policy TD to converge to V^{π_e}?

Q-Learning: Learning the Optimal State-Action Value

- Just saw how to use off policy TD to evaluate any particular policy π_e
- Can we estimate the value of the optimal policy π^* without knowledge of what π^* is?
- Yes! Q-learning
- Does not require importance sampling
- Key idea: Maintain state-action Q estimates and use to bootstrap– use the value of the best future action
- Recall Sarsa

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t)) \quad (6)$$

- Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t)) \quad (7)$$

Off-Policy Control Using Q-learning

- In the prior slide assumed there was some π_b used to act
- π_b determines the actual rewards received
- Now consider how to improve the behavior policy (policy improvement)
- Let behavior policy π_b be ϵ-greedy with respect to (w.r.t.) current estimate of the optimal $q(s, a)$

Q-Learning with ϵ-greedy Exploration

1: Initialize $Q(s, a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
2: Set π_b to be ϵ-greedy w.r.t. Q
3: **loop**
4: Take $a_t \sim \pi_b(s_t)$ // Sample action from policy
5: Observe (r_t, s_{t+1})
6: Update Q given (s_t, a_t, r_t, s_{t+1}):

7: Perform policy improvement: set π_b to be ϵ-greedy w.r.t. Q
8: $t = t + 1$
9: **end loop**

- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal q?
- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal π^*?

Maximization Bias[39]

- 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
- Assume using 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:

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

Double 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 does this yield an unbiased estimate of the max state-action value?

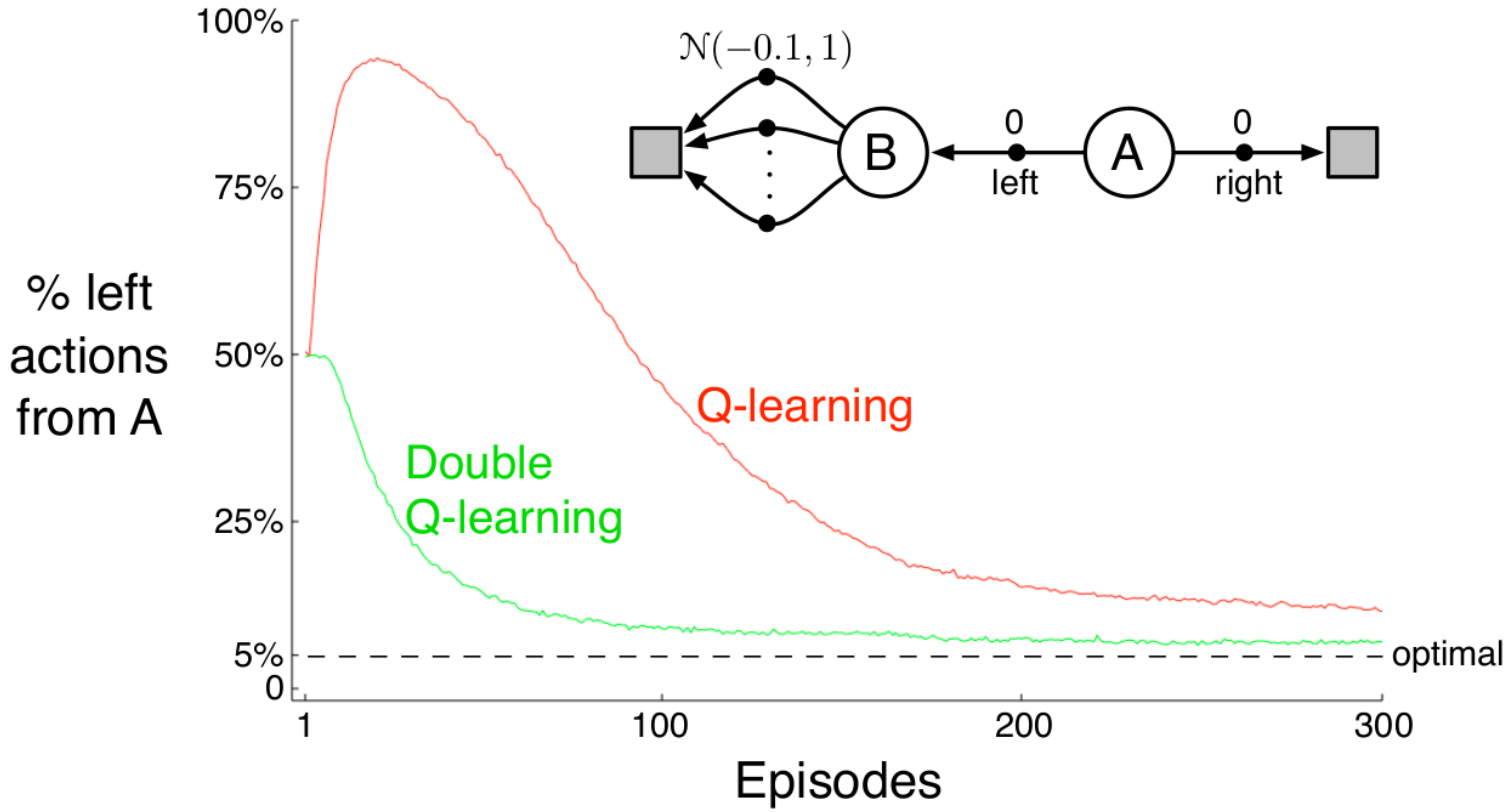
- If acting online, can alternate samples used to update Q_1 and Q_2, using the other to select the action chosen
- Next slides extend to full MDP case (with more than 1 state)

Double Q-Learning

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

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

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.

Lecture 5: Value Function Approximation

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2018

The value function approximation structure for today closely follows much of David Silver's Lecture 6. For additional reading please see SB 2018 Sections 9.3, 9.6-9.7. The deep learning slides come almost exclusively from Ruslan Salakhutdinov's class, and Hugo Larochelle's class (and with thanks to Zico Kolter also for slide inspiration). The slides in my standard style format in the deep learning section are my own.

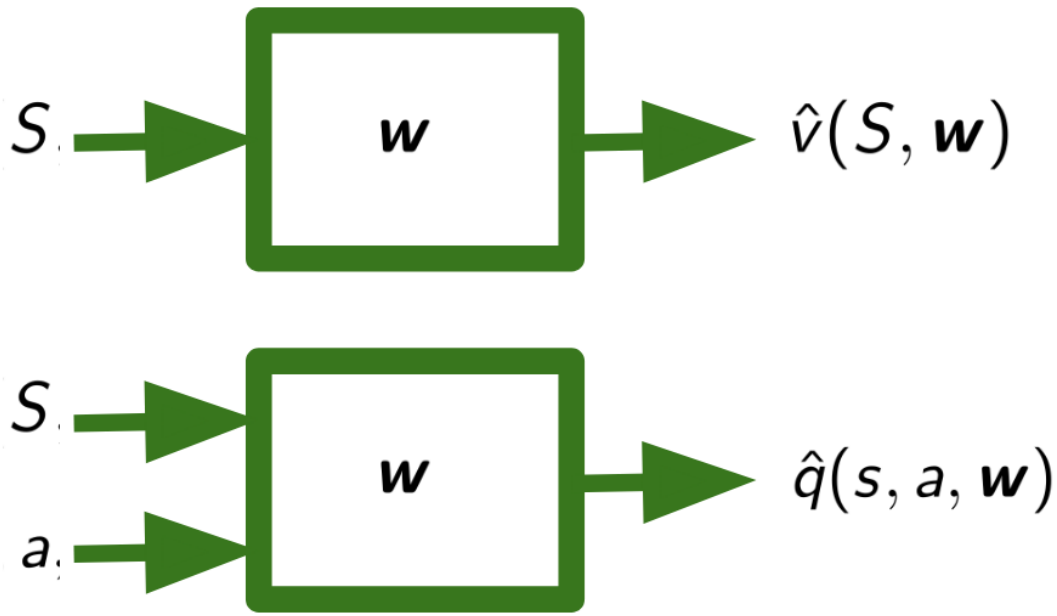
Table of Contents

Last time: Model-Free Control

- Last time: how to learn a good policy from experience
- So far, have been assuming we can represent the value function or state-action value function as a vector
  - Tabular representation
- Many real world problems have enormous state and/or action spaces
- Tabular representation is insufficient

Value Function Approximation (VFA)

- Represent a (state-action/state) value function with a parameterized function instead of a table



Motivation for VFA

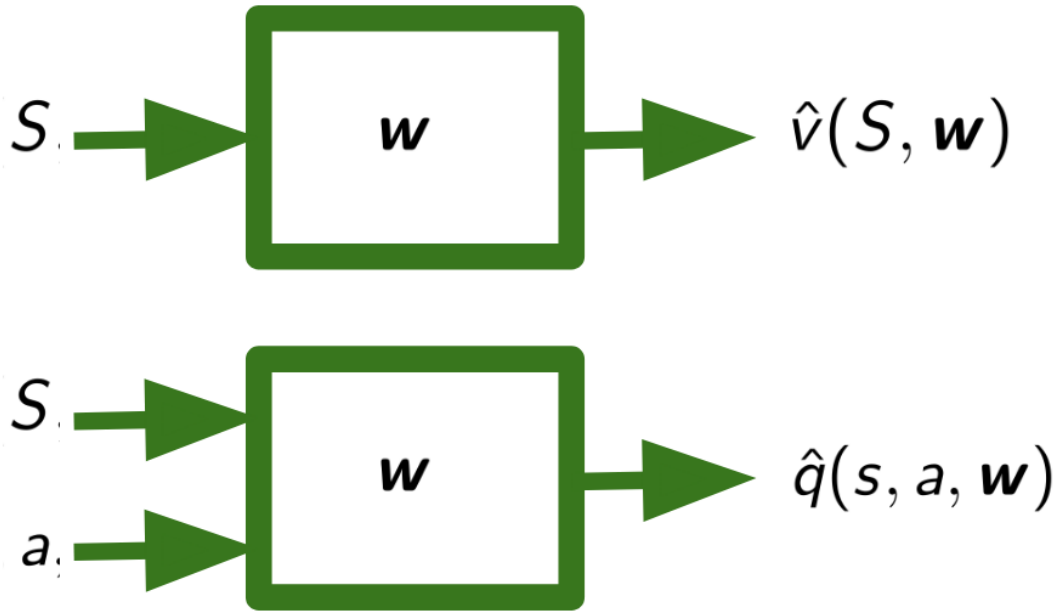
- Don't want to have to explicitly store or learn for every single state a
  - Dynamics or reward model
  - Value
  - State-action value
  - Policy
- Want more compact representation that generalizes across state or states and actions

Benefits of Generalization

- Reduce memory needed to store $(P, R)/V/Q/\pi$
- Reduce computation needed to compute $(P, R)/V/Q/\pi$
- Reduce experience needed to find a good $P, R/V/Q/\pi$

Value Function Approximation (VFA)

- Represent a (state-action/state) value function with a parameterized function instead of a table



- **Which function approximator?**

Function Approximators

- Many possible function approximators including
  - Linear combinations of features
  - Neural networks
  - Decision trees
  - Nearest neighbors
  - Fourier / wavelet bases
- In this class we will focus on function approximators that are differentiable (Why?)
- Two very popular classes of differentiable function approximators
  - Linear feature representations (Today)
  - Neural networks (Today and next lecture)

Review: Gradient Descent

- Consider a function $J(\boldsymbol{w})$ that is a differentiable function of a parameter vector $\boldsymbol{w}$
- Goal is to find parameter $\boldsymbol{w}$ that minimizes J
- The gradient of $J(\boldsymbol{w})$ is

Table of Contents

Value Function Approximation for Policy Evaluation with an Oracle

- First consider if could query any state s and an oracle would return the true value for $v^{\pi}(s)$
- The objective was to find the best approximate representation of v^{π} given a particular parameterized function

Stochastic Gradient Descent

- Goal: Find the parameter vector $\boldsymbol{w}$ that minimizes the loss between a true value function $v_\pi(s)$ and its approximation $\hat{v}$ as represented with a particular function class parameterized by $\boldsymbol{w}$.
- Generally use mean squared error and define the loss as

$$J(\boldsymbol{w}) = \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S, \boldsymbol{w}))^2] \tag{1}$$

- Can use gradient descent to find a local minimum

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \bigtriangledown_{\boldsymbol{w}} J(w) \tag{2}$$

- Stochastic gradient descent (SGD) samples the gradient:

- Expected SGD is the same as the full gradient update

VFA Prediction Without An Oracle

- Don't actually have access to an oracle to tell true $v_\pi(S)$ for any state s
- Now consider how to do value function approximation for prediction / evaluation / policy evaluation without a model
- Note: policy evaluation without a model is sometimes also called **passive reinforcement learning** with value function approximation
  - "passive" because not trying to learn the optimal decision policy

Model Free VFA Prediction / Policy Evaluation

- Recall model-free policy evaluation (Lecture 3)
  - Following a fixed policy π (or had access to prior data)
  - Goal is to estimate V^π and/or Q^π
- Maintained a look up table to store estimates V^π and/or Q^π
- Updated these estimates after each episode (Monte Carlo methods) or after each step (TD methods)
- **Now: in value function approximation, change the estimate update step to include fitting the function approximator**

Feature Vectors

- Use a feature vector to represent a state

$$x(s) = \begin{pmatrix} x_1(s) \\ x_2(s) \\ \dots \\ x_n(s) \end{pmatrix} \tag{3}$$

Linear Value Function Approximation for Prediction With An Oracle

- Represent a value function (or state-action value function) for a particular policy with a weighted linear combination of features

$$\hat{v}(S, \boldsymbol{w}) = \sum_{j=1}^{n} x_j(S) w_j = \boldsymbol{x}(S)^T (w)$$

- Objective function is

$$J(\boldsymbol{w}) = \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S, \boldsymbol{w}))^2]$$

- Recall weight update is

$$\Delta(\boldsymbol{w}) = -\frac{1}{2}\alpha \bigtriangledown_{\boldsymbol{w}} J(w) \tag{4}$$

- Update is:

- Update = step-size $\times$ prediction error $\times$ feature value

Monte Carlo Value Function Approximation

- Return G_t is an unbiased but noisy sample of the true expected return $v_\pi(S_t)$
- Therefore can reduce MC VFA to doing supervised learning on a set of (state,return) pairs: $< S_1, G_1 >, < S_2, G_2 >, \ldots, < S_T, G_T >$
  - Susbstituting $G_t(S_t)$ for the true $v_\pi(S_t)$ when fitting the function approximator
- Concretely when using linear VFA for policy evaluation

$$\begin{aligned} \Delta \boldsymbol{w} &= \alpha(G_t - \hat{v}(S_t, \boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}) && (5) \\ &= \alpha(G_t - \hat{v}(S_t, \boldsymbol{w}))\boldsymbol{x}(S_t) && (6) \end{aligned}$$

- Note: G_t may be a very noisy estimate of true return

MC Linear Value Function Approximation for Policy Evaluation

1: Initialize $\mathbf{w} = \mathbf{0}, Returns(s) = 0 \ \forall (s, a), \ k = 1$
2: **loop**
3: Sample k-th episode $(s_{k1}, a_{k1}, r_{k1}, s_{k2}, \ldots, s_{k,L_k})$ given π
4: **for** $t = 1, \ldots, L_k$ **do**
5: **if** First visit to (s) in episode k **then**
6: Append $\sum_{j=t}^{L_k} r_{kj}$ to $Returns(s_t)$
7: Update weights
8: **end if**
9: **end for**
10: $k = k + 1$
11: **end loop**

Recall: Temporal Difference (TD(0)) Learning with a Look up Table

- Uses bootstrapping and sampling to approximate V^π
- Updates $V^\pi(s)$ after each transition (s, a, r, s'):

$$V^\pi(s) = V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s)) \tag{7}$$

- Target is $r + \gamma V^\pi(s')$, a biased estimate of the true value $v^\pi(s)$
- Look up table represents value for each state with a separate table entry

Temporal Difference (TD(0)) Learning with Value Function Approximation

- Uses bootstrapping and sampling to approximate true v^π
- Updates estimate $V^\pi(s)$ after each transition (s, a, r, s'):

$$V^\pi(s) = V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s)) \quad (8)$$

- Target is $r + \gamma V^\pi(s')$, a biased estimate of of the true value $v^\pi(s)$
- In value function approximation, target is $r + \gamma \hat{v}^\pi(s')$, a biased and approximated estimate of of the true value $v^\pi(s)$
- 3 forms of approximation:

Temporal Difference (TD(0)) Learning with Value Function Approximation

- In value function approximation, target is $r + \gamma \hat{v}^{\pi}(s')$, a biased and approximated estimate of of the true value $v^{\pi}(s)$
- Supervised learning on a different set of data pairs: $< S_1, r_1 + \gamma \hat{v}^{\pi}(S_2, \boldsymbol{w}) >, < S_2, r_2 + \gamma \hat{v}(S_3, \boldsymbol{w}) >, \ldots$

Temporal Difference (TD(0)) Learning with Value Function Approximation

- In value function approximation, target is $r + \gamma \hat{v}^{\pi}(s')$, a biased and approximated estimate of of the true value $v^{\pi}(s)$
- Supervised learning on a different set of data pairs: $< S_1, r_1 + \gamma \hat{v}^{\pi}(S_2, \boldsymbol{w}) >, < S_2, r_2 + \gamma \hat{v}(S_3, \boldsymbol{w}) >, \ldots$
- In linear TD(0)

$$\begin{aligned} \Delta \boldsymbol{w} &= \alpha(r + \gamma \hat{v}^{\pi}(s', \boldsymbol{w}) - \hat{v}^{\pi}(s, \boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \hat{v}^{\pi}(s, \boldsymbol{w}) && (9) \\ &= \alpha(r + \gamma \hat{v}^{\pi}(s', \boldsymbol{w}) - \hat{v}^{\pi}(s, \boldsymbol{w}))\boldsymbol{x}(s) && (10) \end{aligned}$$

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation[1]

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

$$MSVE(\mathbf{w}) = \sum_{\mathbf{s} \in \mathbf{S}} \mathbf{d}(\mathbf{s})(\mathbf{v}^{\pi}(\mathbf{s}) - \hat{\mathbf{v}}^{\pi}(\mathbf{s}, \mathbf{w}))^{2} \tag{11}$$

- where
  - $d(s)$: stationary distribution of π in the true decision process
  - $v,\hat{}w^{\pi}(s) = \boldsymbol{x}(s)^{T}\boldsymbol{w}$, a linear value function approximation

[1] Tsitsiklis and Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. 1997.https://web.stanford.edu/ bvr/pubs/td.pdf

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation[2]

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

$$MSVE(\mathbf{w}) = \sum_{\mathbf{s} \in \mathbf{S}} \mathbf{d}(\mathbf{s})(\mathbf{v}^{\pi}(\mathbf{s}) - \hat{\mathbf{v}}^{\pi}(\mathbf{s}, \mathbf{w}))^{\mathbf{2}} \tag{12}$$

- where
  - $d(s)$: stationary distribution of π in the true decision process
  - $\hat{v}^{\pi}(s) = \boldsymbol{x}(s)^{T}\boldsymbol{w}$, a linear value function approximation
- Monte Carlo policy evaluation with VFA converges to the weights $\boldsymbol{w}_{MC}$ which has the minimum mean squared error possible:

$$MSVE(\boldsymbol{w}_{MC}) = \min_{w} \sum_{s \in S} d(s)(v^{\pi} * (s) - \hat{v}^{\pi}(s, \boldsymbol{w}))^{2} \tag{13}$$

[2] Tsitsiklis and Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. 1997.https://web.stanford.edu/ bvr/pubs/td.pdf

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation[3]

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

$$MSVE(\mathbf{w}) = \sum_{\mathbf{s} \in \mathbf{S}} \mathbf{d}(\mathbf{s})(\mathbf{v}^{\pi} * (\mathbf{s}) - \hat{\mathbf{v}}^{\pi}(\mathbf{s}, \mathbf{w}))^2 \tag{14}$$

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

$$MSVE(\boldsymbol{w}_{TD}) = \frac{1}{1-\gamma} \min_{w} \sum_{s \in S} d(s)(v^{\pi} * (s) - \hat{v}^{\pi}(s, \boldsymbol{w}))^2 \tag{15}$$

[3] ibed.

Summary: Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation[4]

- Monte Carlo policy evaluation with VFA converges to the weights $\boldsymbol{w}_{MC}$ which has the minimum mean squared error possible:

$$MSVE(\boldsymbol{w}_{MC}) = \min_{w} \sum_{s \in S} d(s)(v^{\pi} * (s) - \hat{v}^{\pi}(s, \boldsymbol{w}))^2 \tag{16}$$

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

$$MSVE(\boldsymbol{w}_{TD}) = \frac{1}{1-\gamma} \min_{w} \sum_{s \in S} d(s)(v^{\pi} * (s) - \hat{v}^{\pi}(s, \boldsymbol{w}))^2 \tag{17}$$

- Check your understanding: if the VFA is a tabular representation (one feature for each state), what is the MSVE for MC and TD?

[4] ibed.

Convergence Rates for Linear Value Function Approximation for Policy Evaluation

- Does TD or MC converge faster to a fixed point?
- Not (to my knowledge) definitively understood
- Practically TD learning often converges faster to its fixed value function approximation point

Table of Contents

Control using Value Function Approximation

- Use value function approximation to represent state-action values $\hat{q}^{\pi}(s, a, \boldsymbol{w}) \approx q^{\pi}$
- Interleave
  - Approximate policy evaluation using value function approximation
  - Perform ϵ-greedy policy improvement

Action-Value Function Approximation with an Oracle

- $\hat{q}^{\pi}(s, a, \boldsymbol{w}) \approx q^{\pi}$
- Minimize the mean-squared error between the true action-value function $q^{\pi}(s, a)$ and the approximate action-value function:

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(q^{\pi}(s, a) - \hat{q}^{\pi}(s, a, \boldsymbol{w}))^2] \tag{18}$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \nabla_{\boldsymbol{w}} J(w) = \mathbb{E}\left[(q^{\pi}(s, a) - \hat{q}^{\pi}(s, a, \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{q}^{\pi}(s, a, \boldsymbol{w})\right] \tag{19}$$

$$\Delta(\boldsymbol{w}) = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(w) \tag{20}$$

- Stochastic gradient descent (SGD) samples the gradient

Linear State Action Value Function Approximation with an Oracle

- Use features to represent both the state and action

$$x(s,a) = \begin{pmatrix} x_1(s,a) \\ x_2(s,a) \\ \ldots \\ x_n(s,a) \end{pmatrix} \tag{21}$$

- Represent state-action value function with a weighted linear combination of features

$$\hat{q}(s,a,\boldsymbol{w}) = x(s,a)^T \boldsymbol{w} = \sum_{j=1}^{n} x_j(s,a) w_j \tag{22}$$

- Stochastic gradient descent update:

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \mathbb{E}_\pi[(q^\pi(s,a) - \hat{q}^\pi(s,a,\boldsymbol{w}))^2] \tag{23}$$

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

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{q}(s_t, a_t, \boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \hat{q}(s_t, a_t, \boldsymbol{w}) \tag{24}$$

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

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{q}(s', a', \boldsymbol{w}) - \hat{q}(s, a, \boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) \tag{25}$$

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

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{q}(s_t, a_t, \boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \hat{q}(s_t, a_t, \boldsymbol{w}) \tag{26}$$

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

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{q}(s', a', \boldsymbol{w}) - \hat{q}(s, a, \boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) \tag{27}$$

- 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

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}) - \hat{q}(s, a, \boldsymbol{w})) \bigtriangledown_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) \tag{28}$$

Convergence of TD Methods with VFA

- TD with value function approximation is not following the gradient of an objective function
- 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

Lecture 6: CNNs and Deep Q Learning [2]

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2018

[2] With many slides for DQN from David Silver and Ruslan Salakhutdinov and some vision slides from Gianni Di Caro and images from Stanford CS231n, http://cs231n.github.io/convolutional-networks/

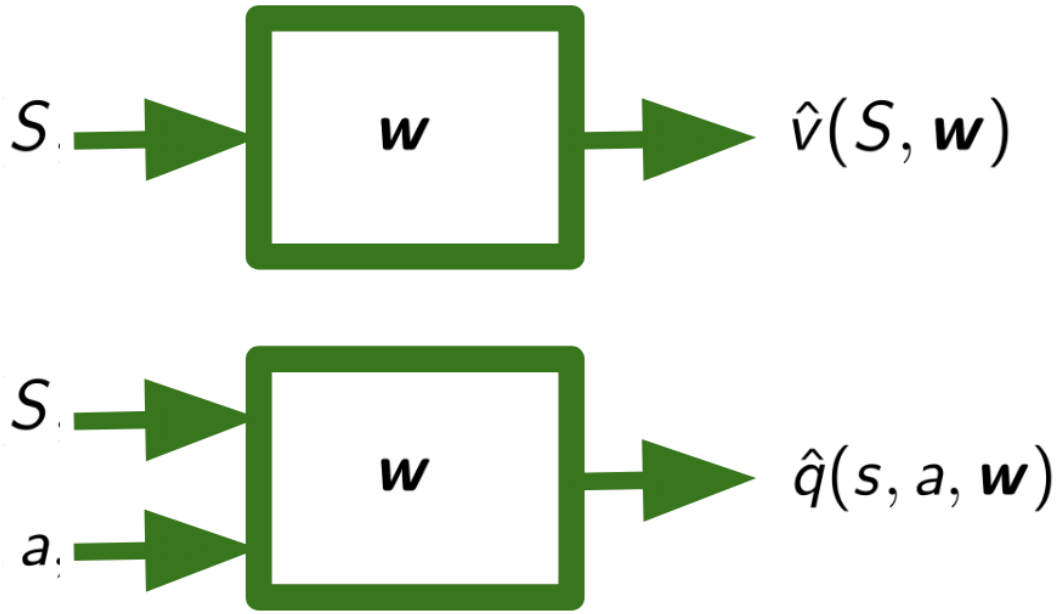
Table of Contents

Class Structure

- Last time: Value function approximation and deep learning
- This time: Convolutional neural networks and deep RL
- Next time: Imitation learning

Generalization

- Want to be able use reinforcement learning to tackle self-driving cars, Atari, consumer marketing, healthcare, education
- Most of these domains have enormous state and/or action spaces
- Requires representations (of models / state-action values / values / policies) that can generalize across states and/or actions
- Represent a (state-action/state) value function with a parameterized function instead of a table



Recall: The Benefit of Deep Neural Network Approximators

- Linear value function approximators assume value function is a weighted combination of a set of features, where each feature a function of the state
- Linear VFA often work well given the right set of features
- But can require carefully hand designing that feature set
- An alternative is to use a much richer function approximation class that is able to directly go from states without requiring an explicit specification of features
- Local representations including Kernel based approaches have some appealing properties (including convergence results under certain cases) but can't typically scale well to enormous spaces and datasets
- Alternative: use deep neural networks
  - Uses distributed representations instead of local representations
  - Universal function approximator
  - Can potentially need exponentially less nodes/parameters (compared to a shallow net) to represent the same function
- Last time discussed basic feedforward deep networks

Table of Contents

Table of Contents

Generalization

- Using function approximation to help scale up to making decisions in really large domains

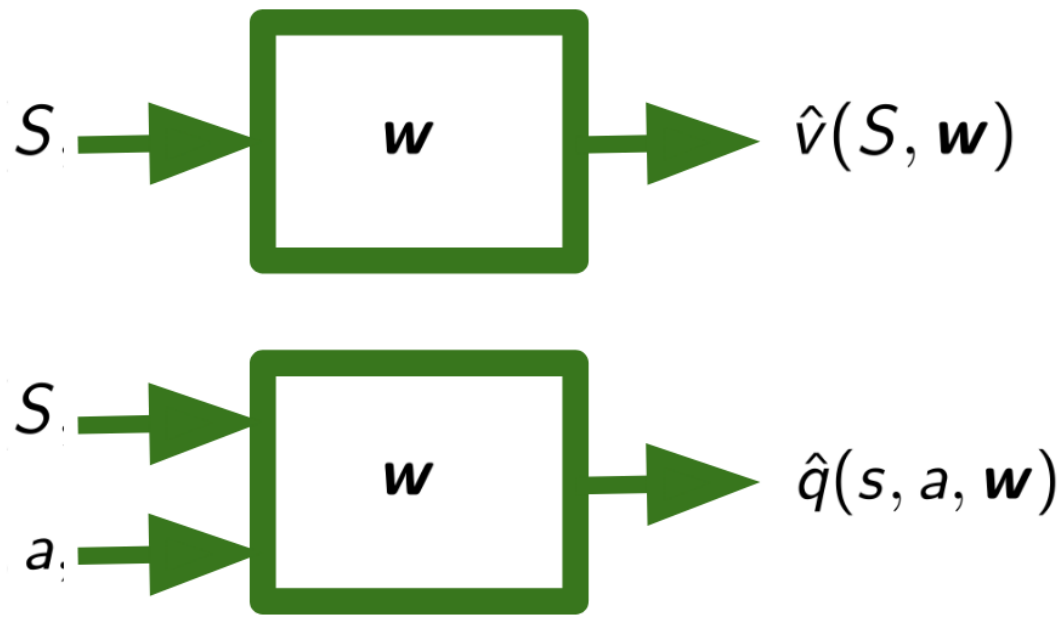
Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimize loss function by stochastic gradient descent (SGD)

Deep Q-Networks (DQNs)

- Represent value function by Q-network with weights $\boldsymbol{w}$

$$\hat{q}(s, a, \boldsymbol{w}) \approx q(s, a) \tag{1}$$



Recall: Action-Value Function Approximation with an Oracle

- $\hat{q}^{\pi}(s, a, w) \approx q^{\pi}$
- Minimize the mean-squared error between the true action-value function $q^{\pi}(s, a)$ and the approximate action-value function:

$$J(w) = \mathbb{E}_{\pi}[(q^{\pi}(s, a) - \hat{q}^{\pi}(s, a, w))^2] \tag{2}$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \bigtriangledown w \, J(w) = \mathbb{E}\left[(q^{\pi}(s, a) - \hat{q}^{\pi}(s, a, w)) \bigtriangledown_w \hat{q}^{\pi}(s, a, w)\right] \tag{3}$$

$$\Delta(w) = -\frac{1}{2}\alpha \bigtriangledown_w J(w) \tag{4}$$

- Stochastic gradient descent (SGD) samples the gradient

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

$$\Delta w = \alpha(G_t - \hat{q}(s_t, a_t, w)) \bigtriangledown_w \hat{q}(s_t, a_t, w) \tag{5}$$

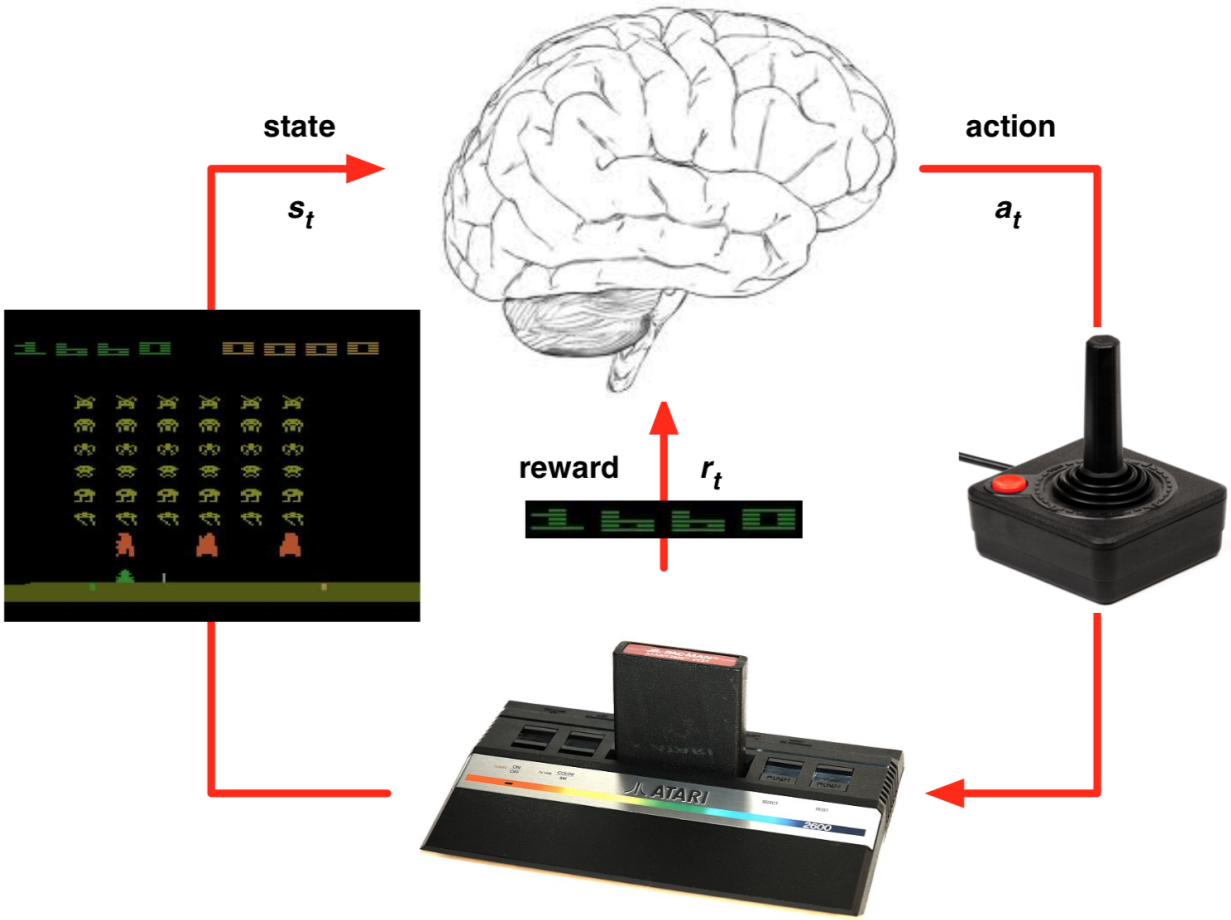
- For SARSA instead use a TD target $r + \gamma \hat{q}(s', a', w)$ which leverages the current function approximation value

$$\Delta w = \alpha(r + \gamma \hat{q}(s', a', w) - \hat{q}(s, a, w)) \bigtriangledown_w \hat{q}(s, a, w) \tag{6}$$

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

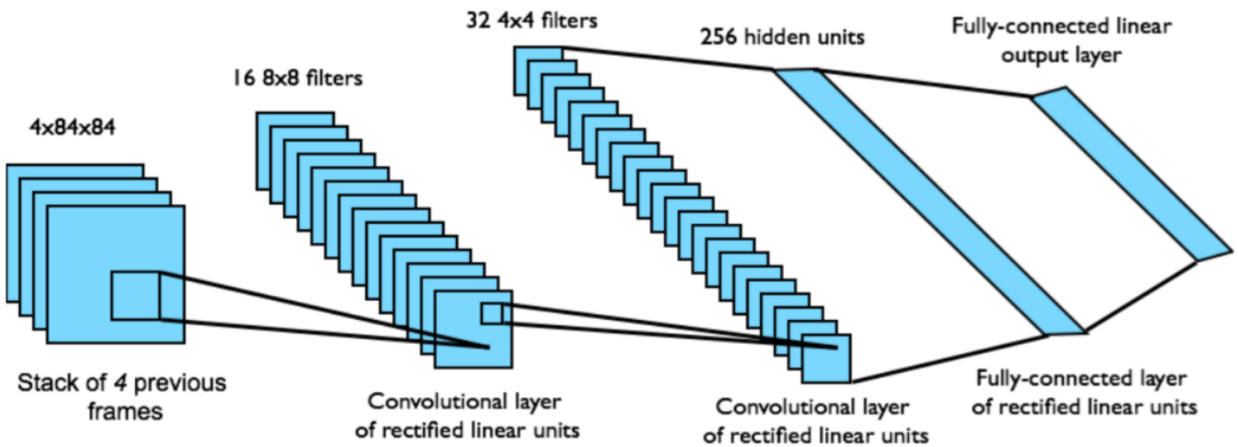
$$\Delta w = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', w) - \hat{q}(s, a, w)) \bigtriangledown_w \hat{q}(s, a, w) \tag{7}$$

Using these ideas to do Deep RL in Atari



DQNs in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

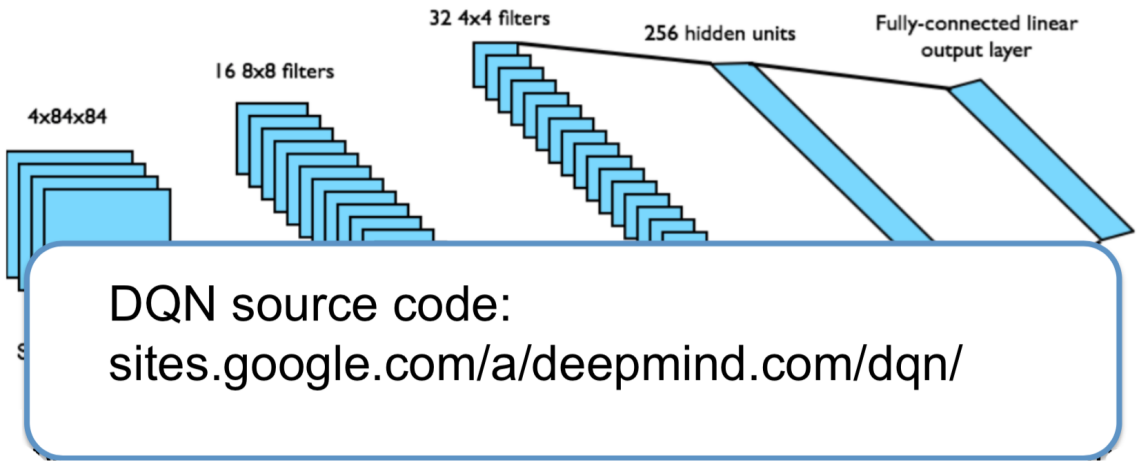


Mnih et.al., Nature, 2014

- Network architecture and hyperparameters fixed across all games

DQNs in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Mnih et.al., Nature, 2014

- Network architecture and hyperparameters fixed across all games

Q-Learning with Value Function Approximation

- Minimize MSE loss by stochastic gradient descent
- Converges to optimal q using table lookup representation
- But Q-learning with VFA can diverge
- Two of the issues causing problems:
  - Correlations between samples
  - Non-stationary targets
- Deep Q-learning (DQN) addresses both of these challenges by
  - Experience replay
  - Fixed Q-targets

DQNs: Experience Replay

- To help remove correlations, store dataset (called a **replay buffer**) $\mathcal{D}$ from prior experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w})$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}) - \hat{q}(s, a, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) \quad (8)$$

DQNs: Experience Replay

- To help remove correlations, store dataset $\mathcal{D}$ from prior experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{q}(s', a', w)$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}) - \hat{q}(s, a, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) \quad (9)$$

- Can treat the target as a scalar, but the weights will get updated on the next round, changing the target value

DQNs: Fixed Q-Targets

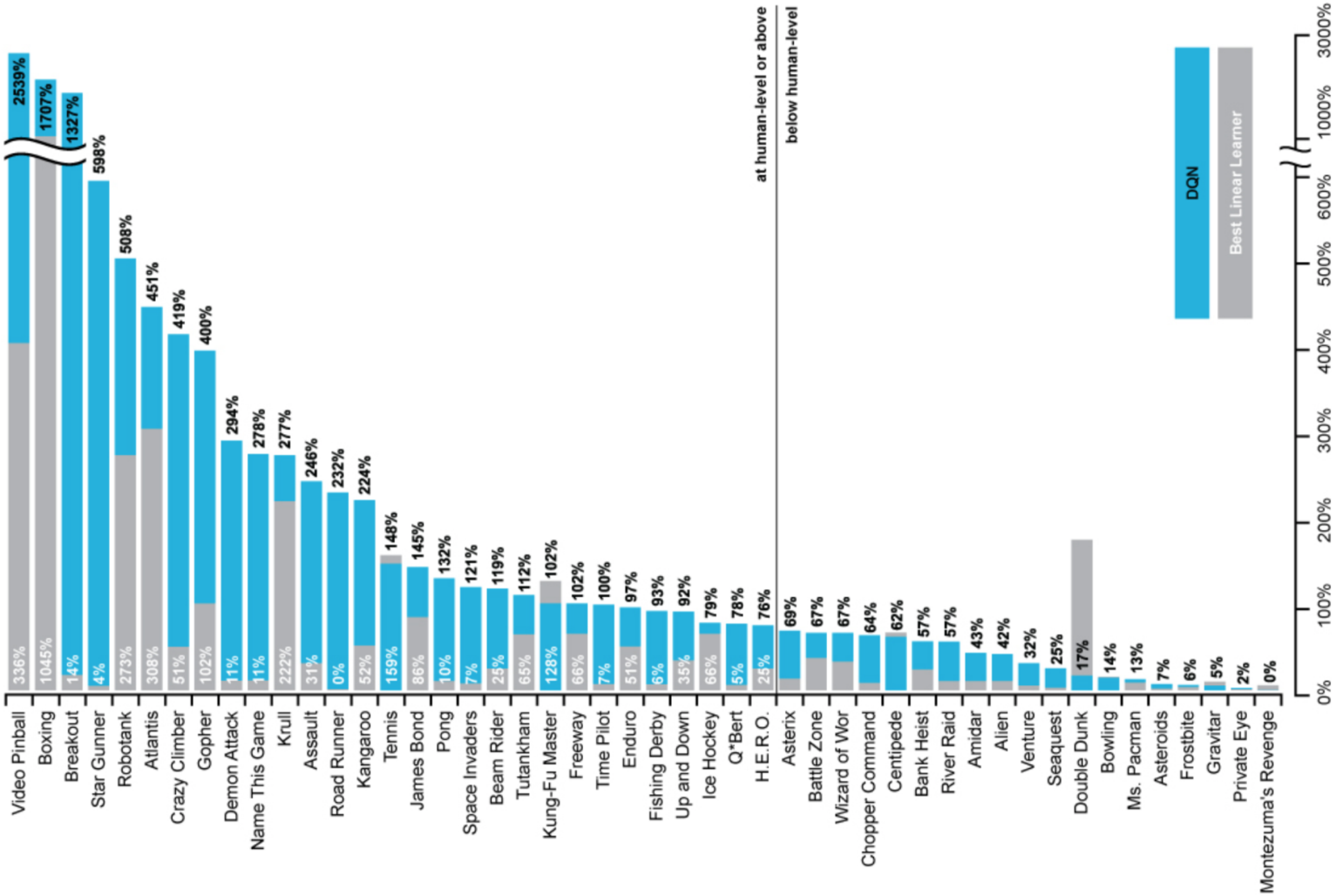
- To help improve stability, fix the **target network** weights used in the target calculation for multiple updates
- Use a different set of weights to compute target than is being updated
- Let parameters $\boldsymbol{w}^-$ be the set of weights used in the target, and $\boldsymbol{w}$ be the weights that are being updated
- Slight change to computation of target value:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}^-)$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}^-) - \hat{q}(s, a, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) \quad (10)$$

DQNs Summary

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters $\boldsymbol{w}^-$
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent

DQN Results in Atari



Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network	DQN w/ fixed Q	DQN w/ replay	DQN w/replay and fixed Q
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Seaquest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089

- Replay is **hugely** important item Why? Beyond helping with correlation between samples, what does replaying do?

Deep RL

- Success in Atari has lead to huge excitement in using deep neural networks to do value function approximation in RL
- Some immediate improvements (many others!)
  - Double DQN
  - Dueling DQN (best paper ICML 2016)

Double DQN

- Recall maximization bias challenge
  - Max of the estimated state-action values can be a biased estimate of the max
- Double Q-learning

Recall: Double Q-Learning

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

$$Q_1(s_t,a_t) \leftarrow Q_1(s_t,a_t) + \alpha(r_t + Q_1(s_{t+1}, \arg\max_{a'} Q_2(s_{t+1},a')) - Q_1(s_t,a_t)) \tag{11}$$

6: **else**

$$Q_2(s_t,a_t) \leftarrow Q_2(s_t,a_t) + \alpha(r_t + Q_2(s_{t+1}, \arg\max_{a'} Q_1(s_{t+1},a')) - Q_2(s_t,a_t))$$

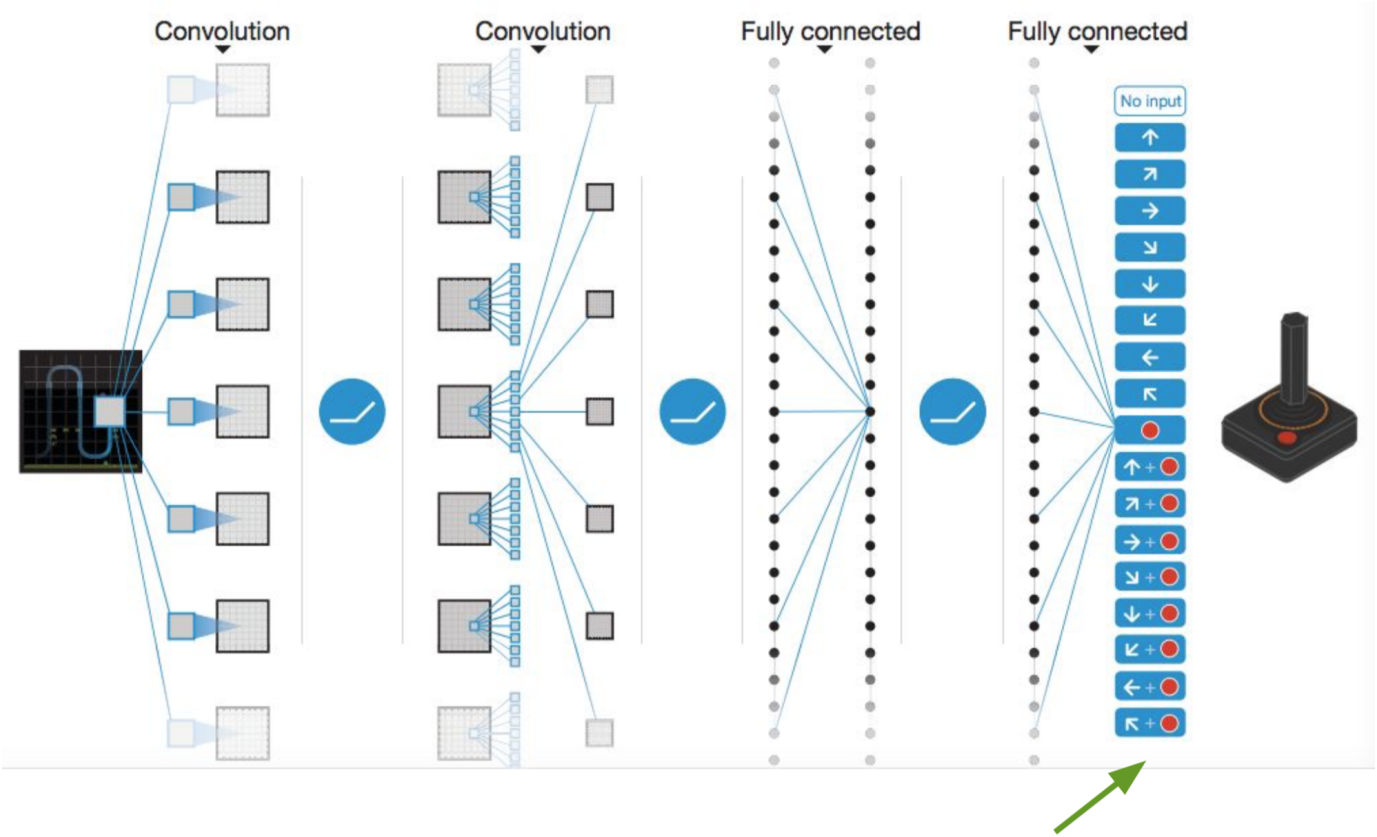
7: **end if**
8: $t = t+1$
9: **end loop**

Double DQN

- Extend this idea to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^-$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{q}(\underbrace{\arg\max_{a'} \hat{q}(s', a', \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}, \boldsymbol{w}^-)}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{q}(s, a, \boldsymbol{w})) \tag{12}$$

Double DQN


Convolution
Convolution
Fully connected
Fully connected
No input
1 network, outputs Q value for each action

Double DQN

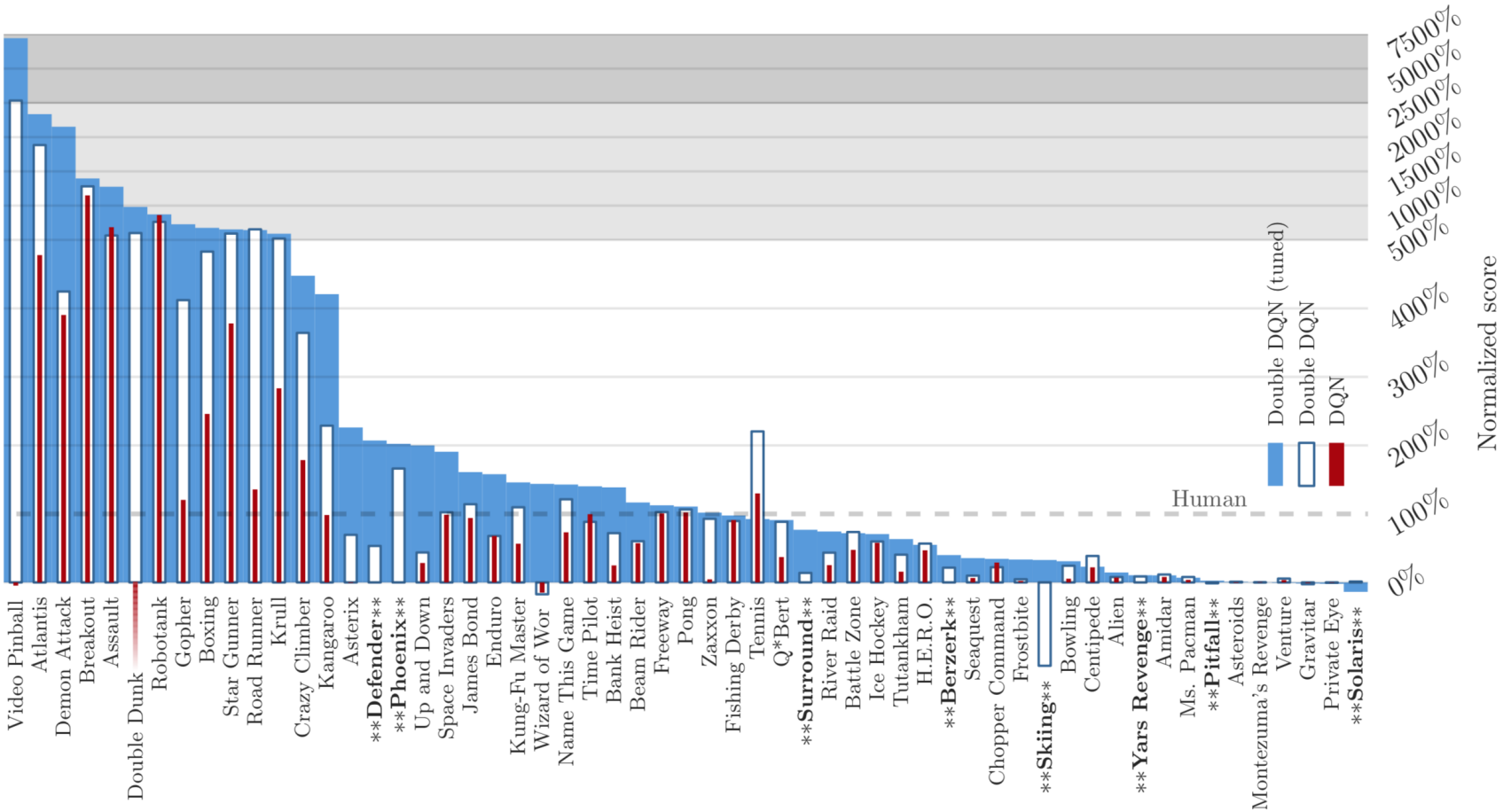


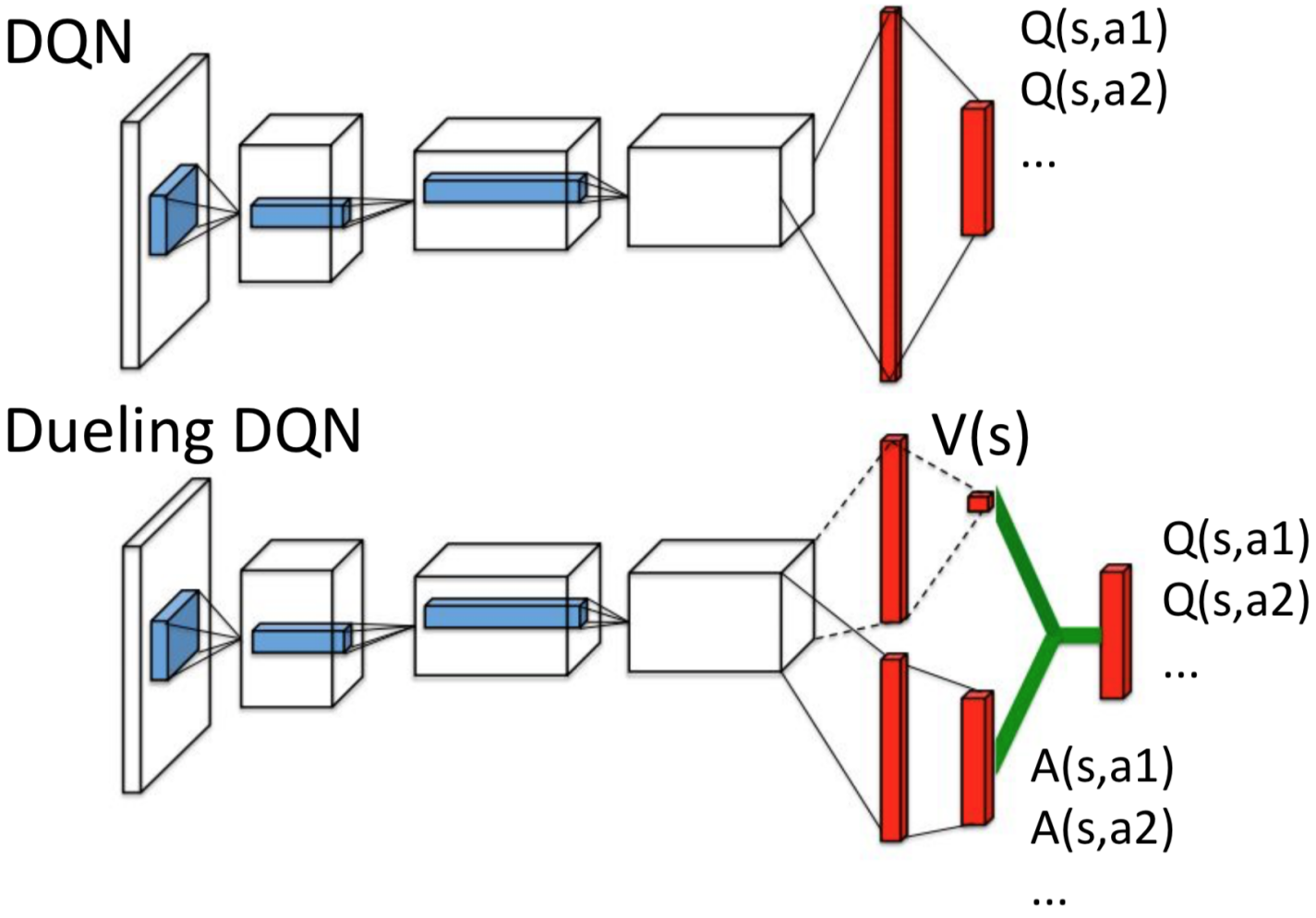
Figure: van Hasselt, Guez, Silver, 2015

Value & Advantage Function

- Intuition: Features need to pay attention to determine value may be different than those need to determine action benefit
- E.g.
  - Game score may be relevant to predicting $V(s)$
  - But not necessarily in indicating relative action values
- Advantage function (Baird 1993)

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

Dueling DQN



Wang et.al., ICML, 2016

Identifiability

- Advantage function

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

- Identifiable?

Identifiability

- Advantage function

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

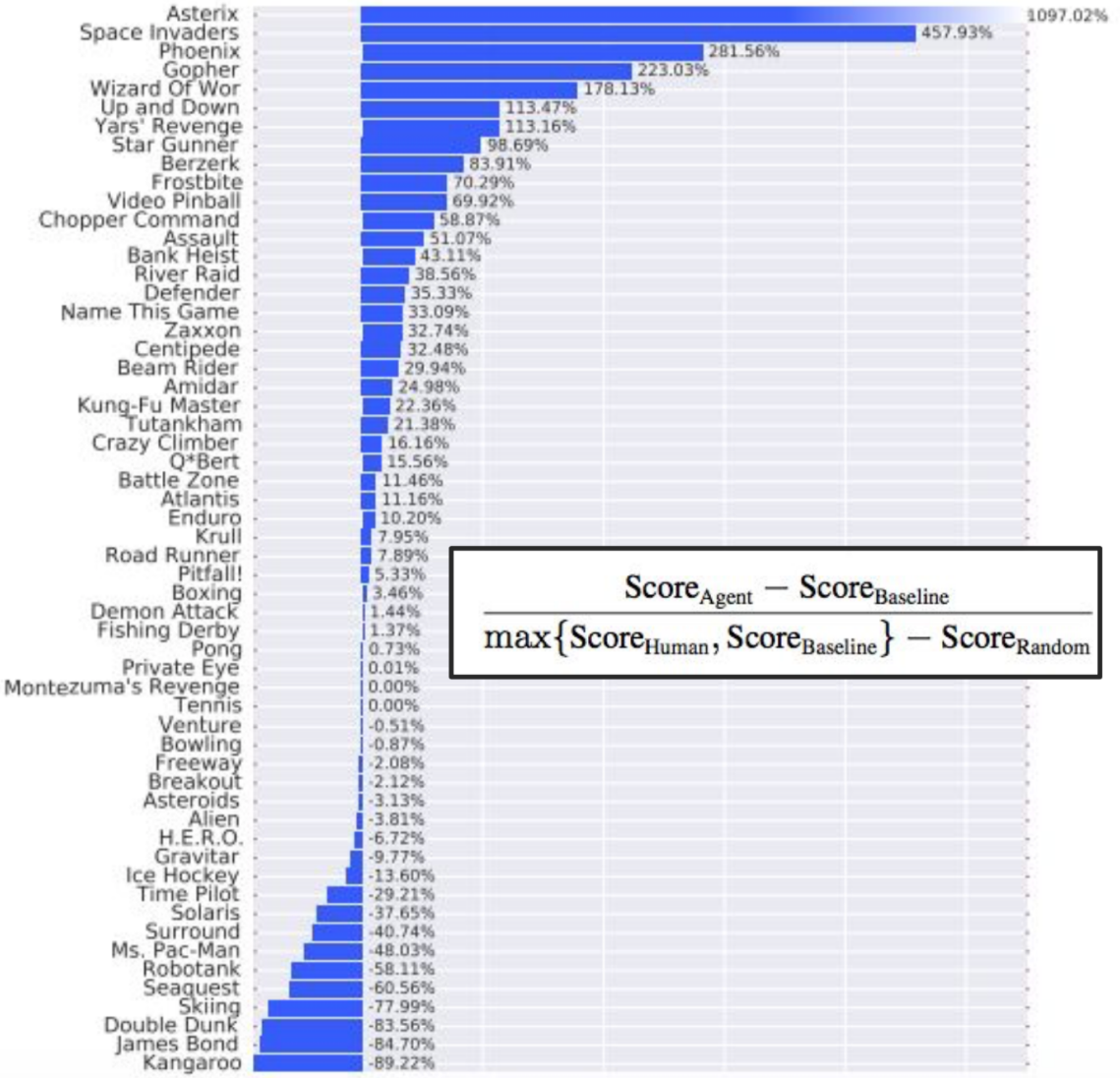
- Unidentifiable
- Option 1: Force $A(s,a) = 0$ if a is action taken

$$\hat{q}(s,a;\boldsymbol{w}) = \hat{v}(s;\boldsymbol{w}) + \left(A(s,a;\boldsymbol{w}) - \max_{a' \in \mathcal{A}} A(s,a';\boldsymbol{w}) \right)$$

- Option 2: Use mean as baseline (more stable)

$$\hat{q}(s,a;\boldsymbol{w}) = \hat{v}(s;\boldsymbol{w}) + \left(A(s,a;\boldsymbol{w}) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\boldsymbol{w}) \right)$$

V.S. DDQN with Prioritized Replay



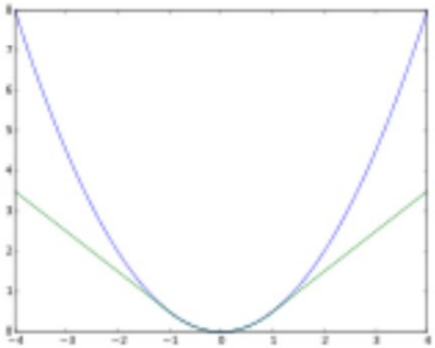
Practical Tips for DQN on Atari (from J. Schulman)

- DQN is more reliable on some Atari tasks than others. Pong is a reliable task: if it doesn't achieve good scores, something is wrong
- Large replay buffers improve robustness of DQN, and memory efficiency is key
  - Use uint8 images, don't duplicate data
- Be patient. DQN converges slowly—for ATARI it's often necessary to wait for 10-40M frames (couple of hours to a day of training on GPU) to see results significantly better than random policy
- In our Stanford class: Debug implementation on small test environment

Practical Tips for DQN on Atari (from J. Schulman) cont.

- Try Huber loss on Bellman error

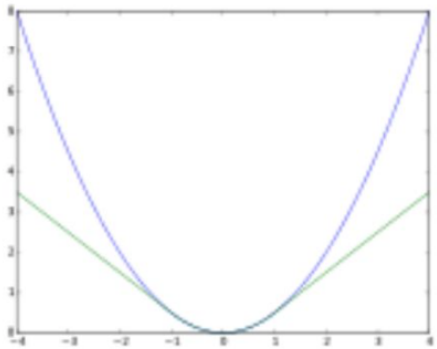
$$L(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \leq \delta \\ \delta|x| - \frac{\delta^2}{2} & \text{otherwise} \end{cases}$$



Practical Tips for DQN on Atari (from J. Schulman) cont.

- Try Huber loss on Bellman error

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



- Consider trying Double DQN—significant improvement from 3-line change in Tensorflow.
- To test out your data pre-processing, try your own skills at navigating the environment based on processed frames
- Always run at least two different seeds when experimenting
- Learning rate scheduling is beneficial. Try high learning rates in initial exploration period
- Try non-standard exploration schedules

Table of Contents