Lecture 5: Value Function Approximation

Emma Brunskill

CS234 Reinforcement Learning.

The value function approximation structure for today closely follows much of David Silver's Lecture 6.

Refresh Your Knowledge 4

- The basic idea of TD methods are to make state-next state pairs fit the constraints of the Bellman equation on average (question by: Phil Thomas)
  1. True
  2. False
  3. Not sure
- In tabular MDPs, if using a decision policy that visits all states an infinite number of times, and in each state randomly selects an action, then (select all)
  1. Q-learning will converge to the optimal Q-values
  2. SARSA will converge to the optimal Q-values
  3. Q-learning is learning off-policy
  4. SARSA is learning off-policy
  5. Not sure
- A TD error > 0 can occur even if the current $V(s)$ is correct $\forall s$: [select all]
  1. False
  2. True if the MDP has stochastic state transitions
  3. True if the MDP has deterministic state transitions
  4. Not sure

Refresh Your Knowledge 4

- The basic idea of TD methods are to make state-next state pairs fit the constraints of the Bellman equation on average (question by: Phil Thomas)
  1. True (True)
  2. False
  3. Not sure
- In tabular MDPs, if using a decision poilcy that visits all states an infinite number of times, and in each state randomly selects an action, then (select all)
  1. Q-learning will converge to the optimal Q-values (True)
  2. SARSA will converge to the optimal Q-values (False)
  3. Q-learning is learning off-policy (True)
  4. SARSA is learning off-policy (False)
- A TD error > 0 can occur even if the current $V(s)$ is correct $\forall s$: [select all]
  1. False
  2. True if the MDP has stochastic state transitions (True)
  3. True if the MDP has deterministic state transitions (False)
  4. Not sure

$r + \gamma V^{\pi}(s') - V^{\pi}(s)$

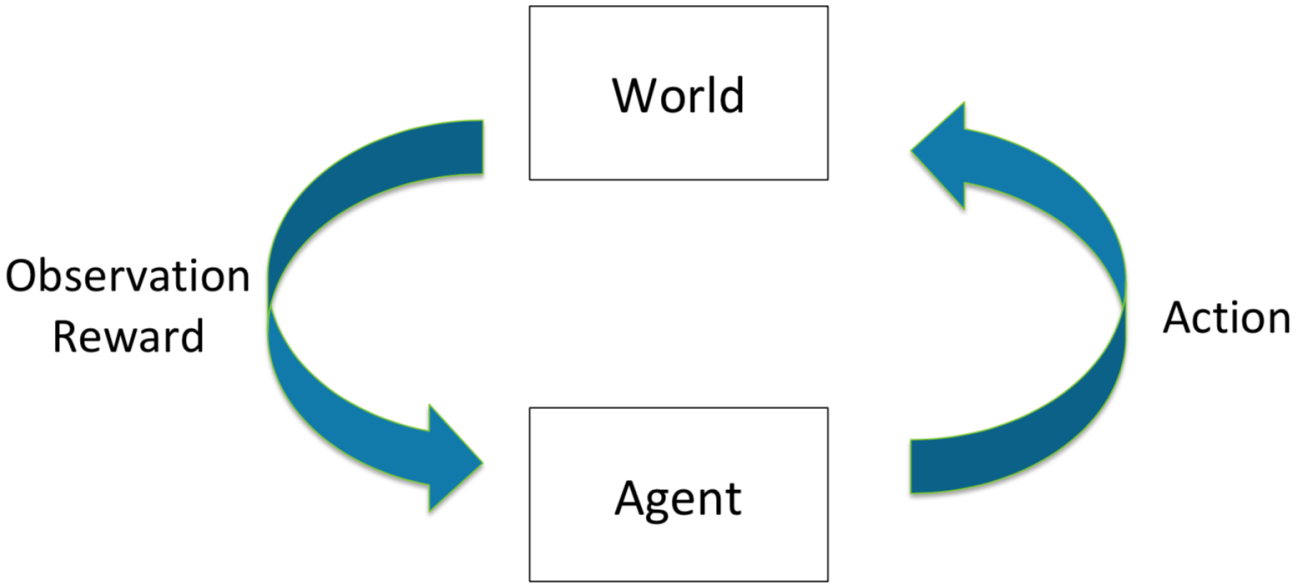
Class Structure

- Last time: Control (making decisions) without a model of how the world works
- **This time: Linear value function approximation**
- Next time: Deep reinforcement learning

Outline for Today

- Value function approximation
- Monte Carlo policy evaluation with linear function approximation
- TD policy evaluation with linear function approximation
- Control methods with linear value function approximation

Reinforcement Learning



- Goal: Learn to select actions to maximize total expected future reward

Last Time: Tabular Representations for 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/ matrix
  - Tabular representation
- Many real world problems have enormous state and/or action spaces
- Tabular representation is insufficient

Motivation for Function Approximation

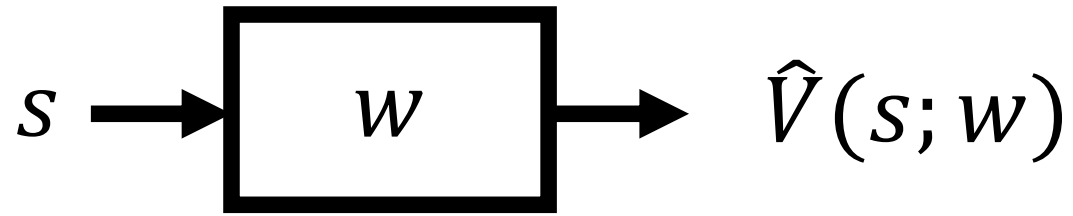
- Don't want to have to explicitly store or learn for every single state a
  - Dynamics or reward model
  - Value
  - State-action value
  - Policy
- Want more compact representation that generalizes across state or states and actions

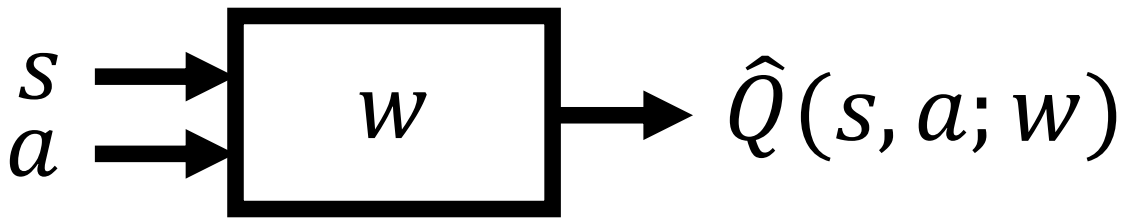
Benefits of Function Approximation

- 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 (Next lecture)

Outline for the Rest of Today: Policy Evaluation to Control

- Given known dynamics and reward models, and a tabular representation
  - Discussed how to do policy evaluation and then control (value iteration and policy iteration)
- Given no models, and a tabular representation
  - Discussed how to do policy evaluation (MC/TD) and then control (MC, SARSA, Q-learning)
- **Given no models, and function approximation**
  - **Today will discuss how to do policy evaluation and then control**

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

Value Function Approximation for Policy Evaluation with an Oracle

- First assume we could query any state s and an oracle would return the true value for $V^{\pi}(s)$
- Similar to supervised learning: assume given $(s, V^{\pi}(s))$ pairs
- The objective is to find the best approximate representation of V^{π} given a particular parameterized function $\hat{V}(s; w)$

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 $\hat{V}(s; \boldsymbol{w})$ as represented with a particular function class parameterized by $\boldsymbol{w}$.
- Generally use mean squared error and define the loss as

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(V^{\pi}(s) - \hat{V}(s; \boldsymbol{w}))^2]$$

- Can use gradient descent to find a local minimum

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

- Stochastic gradient descent (SGD) uses a finite number of (often one) samples to compute an approximate gradient:

- Expected SGD is the same as the full gradient update

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 $\hat{V}(s; \boldsymbol{w})$ as represented with a particular function class parameterized by $\boldsymbol{w}$.
- Generally use mean squared error and define the loss as

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(V^{\pi}(s) - \hat{V}(s; \boldsymbol{w}))^2]$$

- Can use gradient descent to find a local minimum

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

- Stochastic gradient descent (SGD) uses a finite number of (often one) samples to compute an approximate gradient:

$$\begin{aligned} \Delta_w J(w) &= \Delta_w E_{\pi}[V^{\pi}(s) - \hat{V}(s; w)]^2 \\ &= E_{\pi}[2(V^{\pi}(s) - \hat{V}(s; w)]\Delta_w \hat{V}(s, w) \end{aligned}$$

- Expected SGD is the same as the full gradient update

Model Free VFA Policy Evaluation

- Don't actually have access to an oracle to tell true $V^{\pi}(s)$ for any state s
- Now consider how to do model-free value function approximation for prediction / evaluation / policy evaluation without a model

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 lookup 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**

Outline for Today

- Value function approximation
- **Monte Carlo policy evaluation with linear function approximation**
- TD policy evaluation with linear function approximation
- Control methods with linear value function approximation

Feature Vectors

- Use a feature vector to represent a state s

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

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

$$\hat{V}(s; \boldsymbol{w}) = \sum_{j=1}^{n} x_j(s) w_j = \boldsymbol{x}(s)^T \boldsymbol{w}$$

- Objective function is

$$J(\boldsymbol{w}) = \mathbb{E}_\pi[(V^\pi(s) - \hat{V}(s; \boldsymbol{w}))^2]$$

- Recall weight update is

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

- 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: $\langle s_1, G_1 \rangle, \langle s_2, G_2 \rangle, \ldots, \langle s_T, G_T \rangle$
  - Substitute G_t for the true $V^{\pi}(s_t)$ when fit function approximator

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: $\langle s_1, G_1 \rangle, \langle s_2, G_2 \rangle, \ldots, \langle s_T, G_T \rangle$
  - Substitute G_t for the true $V^{\pi}(s_t)$ when fit function approximator
- Concretely when using linear VFA for policy evaluation

$$\begin{aligned} \Delta \boldsymbol{w} &= \alpha(G_t - \hat{V}(s_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{V}(s_t; \boldsymbol{w}) \\ &= \alpha(G_t - \hat{V}(s_t; \boldsymbol{w}))\boldsymbol{x}(s_t) \\ &= \alpha(G_t - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t) \end{aligned}$$

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

MC Linear Value Function Approximation for Policy Evaluation

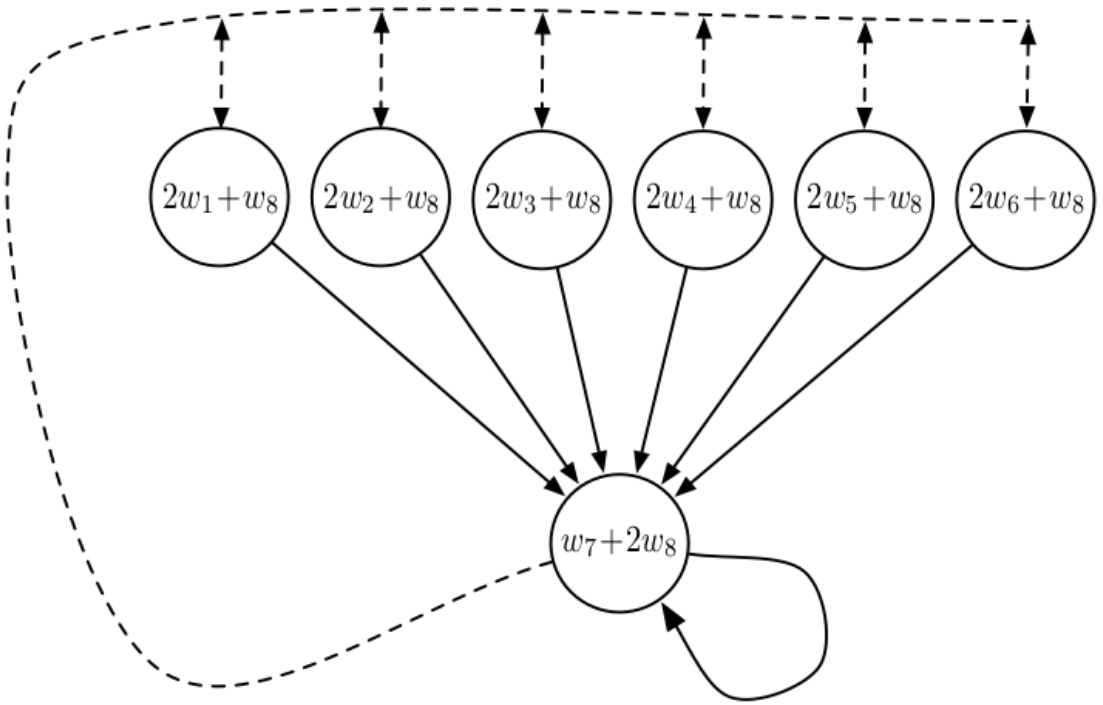
1: Initialize w $= 0$, $k = 1$
2: **loop**
3: Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,L_k})$ given π
4: **for** $t = 1, \ldots, L_k$ **do**
5: **if** First visit to (s) in episode k **then**
6: $G_t(s) = \sum_{j=t}^{L_k} r_{k,j}$
7: Update weights:
8: **end if**
9: **end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

Baird (1995)-Like Example with MC Policy Evaluation[1]

s1= [2 0 0 0 0 0 0 1]
s2 = [0 2 0 0 0 0 0 1]
...
s6 = [0 0 0 0 0 2 0 1]
s7 = [0 0 0 0 0 0 1 2]

w0 = [1 1 1 1 1 1 1 1]
rewards all 0

s7 small prob of going to terminal state



- MC update: $\Delta \boldsymbol{w} = \alpha(G_t - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t)$
- Small prob s_7 goes to terminal state, $\boldsymbol{x}(s_7)^T = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 2]$

Convergence Guarantees for Linear Value Function Approximation 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:

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

Note that μ(s) on the next slide will require a stationary distribution to compute, but the stationary distribution d(s) is for an infinite horizon when using Temporal Difference 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

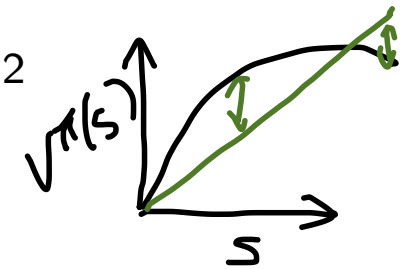
$$MSVE_{\mu}(\boldsymbol{w}) = \sum_{s \in S} \mu(s)(V^{\pi}(s) - \hat{V}^{\pi}(s; \boldsymbol{w}))^2$$

- 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

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



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

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

Today: Focus on Generalization using Linear Value Function

- Preliminaries
- Monte Carlo policy evaluation with linear function approximation
- **TD policy evaluation with linear function approximation**
- Control methods with linear value function approximation

Recall: Temporal Difference Learning w/ Lookup Table

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

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

- Target is $r + \gamma V^\pi(s')$, a biased estimate of the true value $V^\pi(s)$
- Represent 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'):

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

- Target is $r + \gamma V^\pi(s')$, a biased estimate of the true value $V^\pi(s)$
- In value function approximation, target is $r + \gamma \hat{V}^\pi(s'; \boldsymbol{w})$, a biased and approximated estimate of the true value $V^\pi(s)$
- 3 forms of approximation:
  1. Sampling
  2. Bootstrapping
  3. Value function approximation

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

- In value function approximation, target is $r + \gamma \hat{V}^{\pi}(s'; \boldsymbol{w})$, a biased and approximated estimate of the true value $V^{\pi}(s)$
- Can reduce doing TD(0) learning with value function approximation to supervised learning on a set of data pairs:
  - $\langle s_1, r_1 + \gamma \hat{V}^{\pi}(s_2; \boldsymbol{w}) \rangle, \langle s_2, r_2 + \gamma \hat{V}(s_3; \boldsymbol{w}) \rangle, \ldots$
- Find weights to minimize mean squared error

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(r_j + \gamma \hat{V}^{\pi}(s_{j+1}, \boldsymbol{w}) - \hat{V}(s_j; \boldsymbol{w}))^2]$$

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

- In value function approximation, target is $r + \gamma \hat{V}^{\pi}(s'; \boldsymbol{w})$, a biased and approximated estimate of the true value $V^{\pi}(s)$
- Supervised learning on a different set of data pairs: $\langle s_1, r_1 + \gamma \hat{V}^{\pi}(s_2; \boldsymbol{w}) \rangle, \langle s_2, r_2 + \gamma \hat{V}(s_3; \boldsymbol{w}) \rangle, \ldots$
- In linear TD(0)

$$\begin{aligned} \Delta \boldsymbol{w} &= \alpha(r + \gamma \hat{V}^{\pi}(s'; \boldsymbol{w}) - \hat{V}^{\pi}(s; \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{V}^{\pi}(s; \boldsymbol{w}) \\ &= \alpha(r + \gamma \hat{V}^{\pi}(s'; \boldsymbol{w}) - \hat{V}^{\pi}(s; \boldsymbol{w})) \boldsymbol{x}(s) \\ &= \alpha(r + \gamma \boldsymbol{x}(s')^{T} \boldsymbol{w} - \boldsymbol{x}(s)^{T} \boldsymbol{w}) \boldsymbol{x}(s) \end{aligned}$$

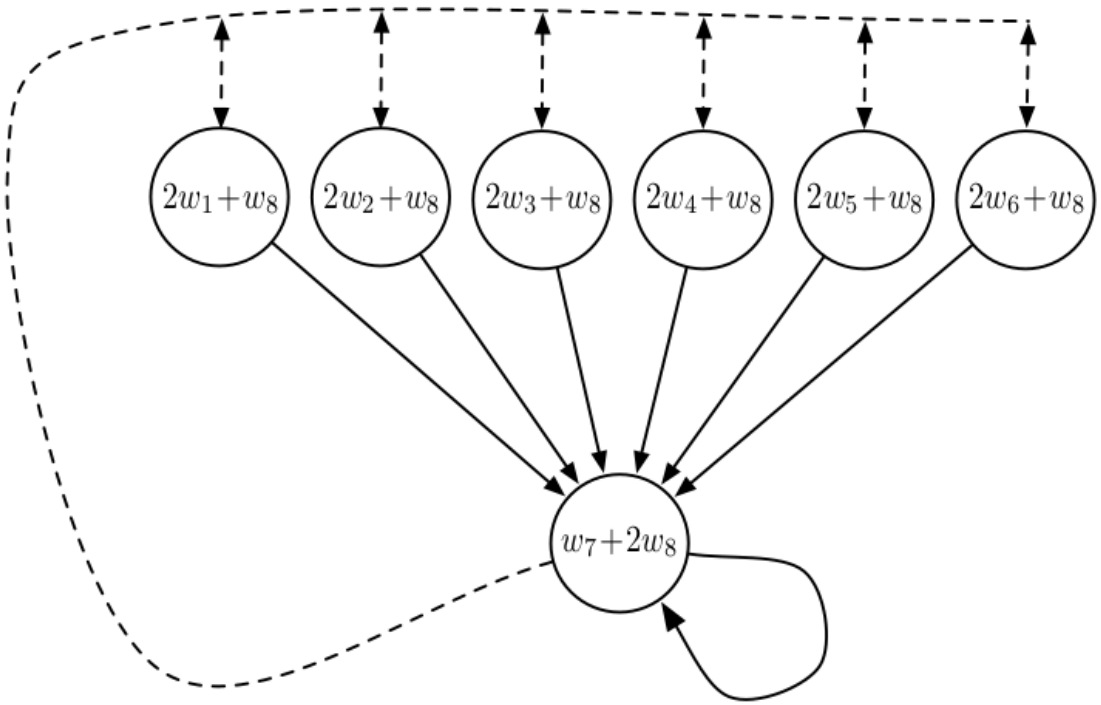
TD(0) Linear Value Function Approximation for Policy Evaluation

1: Initialize $\mathbf{w} = \mathbf{0}$, $k = 1$
2: **loop**
3: Sample tuple (s_k, a_k, r_k, s_{k+1}) given π
4: Update weights:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha(r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w} - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}(s)$$

5: $k = k + 1$
6: **end loop**

Baird Example with TD(0) On Policy Evaluation [1]



- TD update: $\Delta \boldsymbol{w} = \alpha(r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w} - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}(s)$

[1] Figure from Sutton and Barto 2018

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 dynamics model

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

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 as

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

- where
  - $d(s)$: stationary distribution of π in the true decision process
  - $\hat{V}^{\pi}(s; \boldsymbol{w}) = \boldsymbol{x}(s)^T \boldsymbol{w}$, 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:

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

Check Your Understanding: 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: 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:

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

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

1. $\text{MSVE}_d = 0$ for TD (Answer)

Outline for Today

- Value function approximation
- Monte Carlo policy evaluation with linear function approximation
- TD policy evaluation with linear function approximation
- Control methods with linear value function approximation

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
- Can be unstable. Generally involves intersection of the following:
  - Function approximation
  - Bootstrapping
  - **Off-policy learning**

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:

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

- Use stochastic gradient descent to find a local minimum

$$\begin{aligned} -\frac{1}{2}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) &= \mathbb{E}\left[(Q^{\pi}(s, a) - \hat{Q}^{\pi}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}^{\pi}(s, a; \boldsymbol{w})\right] \\ \Delta(\boldsymbol{w}) &= -\frac{1}{2}\alpha\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \end{aligned}$$

- Stochastic gradient descent (SGD) samples the gradient

Check Your Understanding: Predict Control Updates

- The weight update for control for MC and TD-style methods will be near identical to the policy evaluation steps. Try to see if you can predict which are the right weight update equations for the different methods (select all that are true)
- (1) is the SARSA control update
- (2) is the MC control update
- (3) is the Q-learning control update
- (4) is the MC control update
- (5) is the Q-learning control update

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) \quad (1)$$

$$\Delta \boldsymbol{w} = \alpha(G_t + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) \quad (2)$$

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) \quad (3)$$

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}) \quad (4)$$

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{s'} \hat{Q}(s', a; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) \quad (5)$$

Check Your Understanding: Answers

- The weight update for control for MC and TD-style methods will be near identical to the policy evaluation steps. Try to see if you can predict which are the right weight update equations for the different methods.
- (1) is the SARSA control update s a r s' a'
- (3) is the Q-learning control update
- (4) is the MC control update

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) \quad (1)$$

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w}) \quad (3)$$

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}) \quad (4)$$

Linear State Action Value Function Approximation with an Oracle

- Use features to represent both the state and action

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

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

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

- Stochastic gradient descent update:

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

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

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

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

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

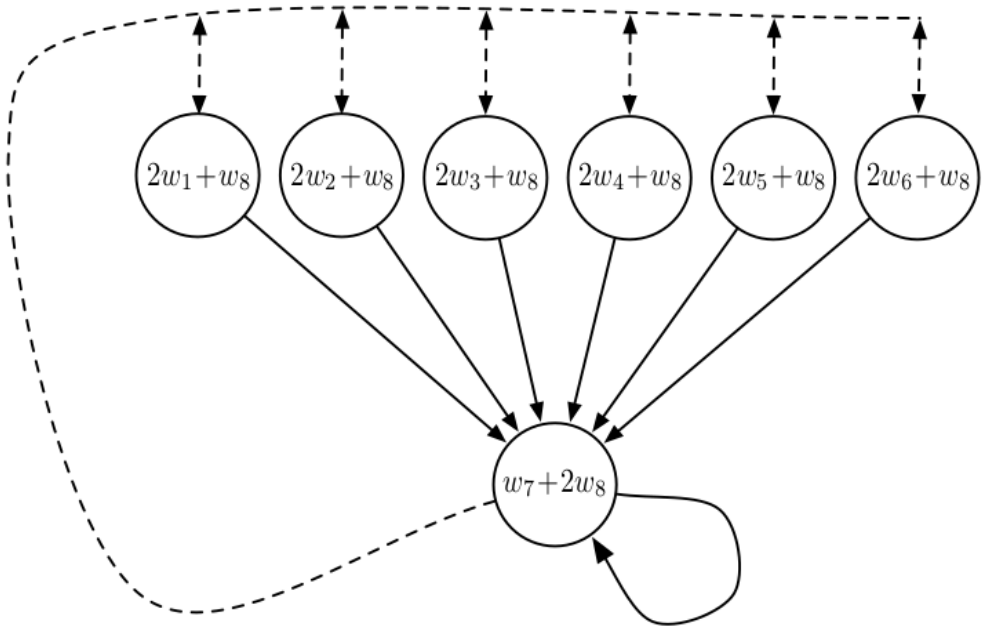
- For Q-learning instead use a TD target $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the max of the current function approximation value

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

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

Challenges of Off Policy Control: Baird Example [1]



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

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

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

$\gamma = 0.99$

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

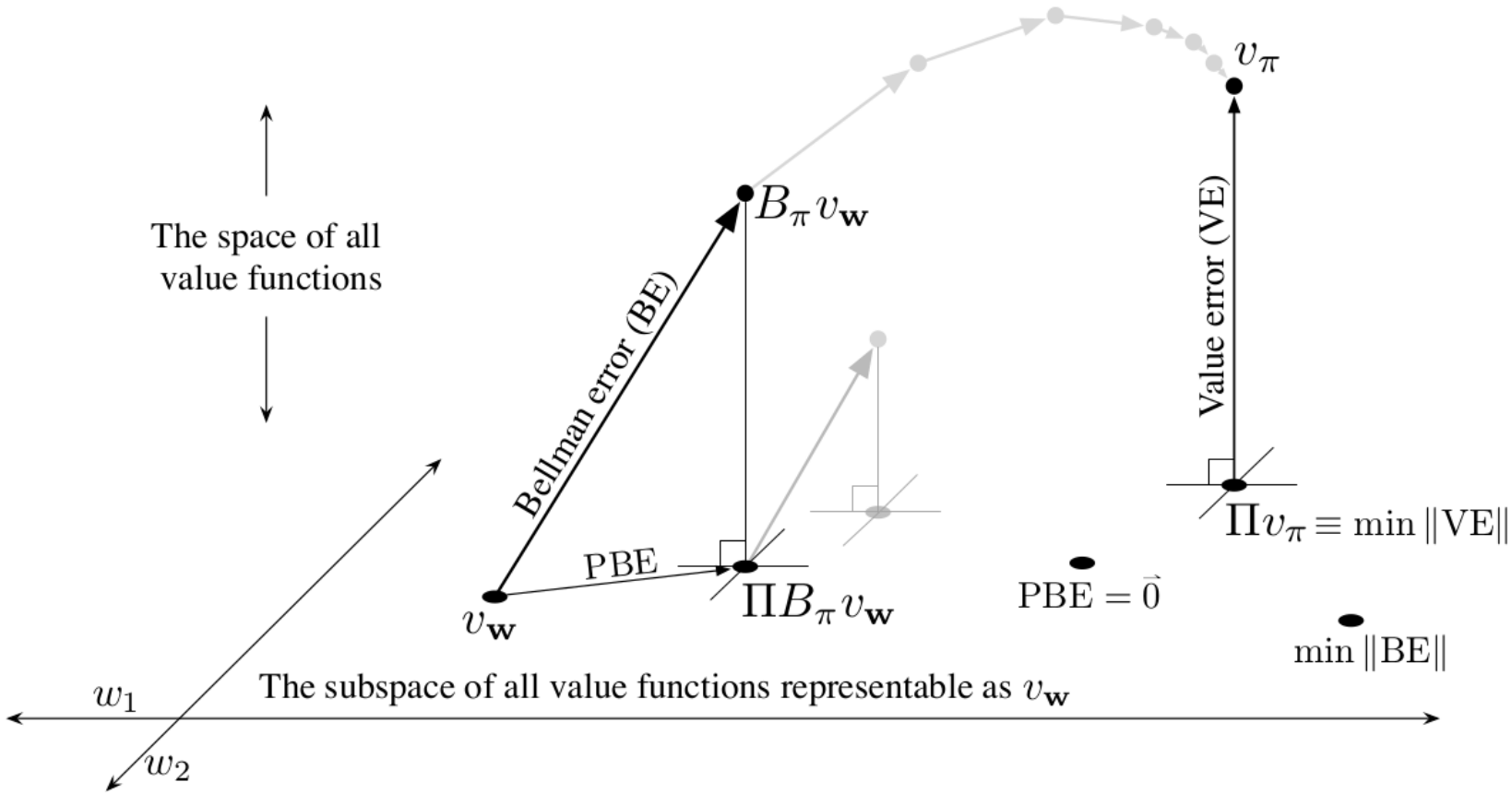
Convergence of Control Methods with VFA

Algorithm	Tabular	Linear VFA	Nonlinear VFA
Monte-Carlo Control			
Sarsa			
Q-learning			

Hot Topic: Off Policy Function Approximation Convergence

- Extensive work in better TD-style algorithms with value function approximation, some with convergence guarantees: see Chp 11 SB
- Exciting recent work on batch RL that can converge with nonlinear VFA (Dai et al. ICML 2018): uses primal dual optimization
- An important issue is not just whether the algorithm converges, but **what** solution it converges too
- Critical choices: **objective function and feature representation**

Linear Value Function Approximation[3]



[3] Figure from Sutton and Barto 2018

What You Should Understand

- Be able to implement TD(0) and MC on policy evaluation with linear value function approximation
- Be able to define what TD(0) and MC on policy evaluation with linear VFA are converging to and when this solution has 0 error and non-zero error.
- Be able to implement Q-learning and SARSA and MC control algorithms
- List the 3 issues that can cause instability and describe the problems qualitatively: function approximation, bootstrapping and off policy learning

Class Structure

- Last time: Control (making decisions) without a model of how the world works
- This time: Value function approximation
- **Next time**: Deep reinforcement learning

Lecture 6: CNNs and Deep Q Learning [1]

Emma Brunskill

CS234 Reinforcement Learning.

[1] 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/

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. $\boldsymbol{w} = \boldsymbol{w} + \alpha(r(s_t) + \gamma \boldsymbol{x}(s_{t+1})^T \boldsymbol{w} - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t)$
  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

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 $\frac{1}{1-\gamma}$. The weights do not depend on the state.

Class Structure

- Last time: Value function approximation
- This time: RL with function approximation, deep RL
- Next time: Deep RL continued

Today

- Value function approximation
- Deep neural networks
- CNNs
- DQN

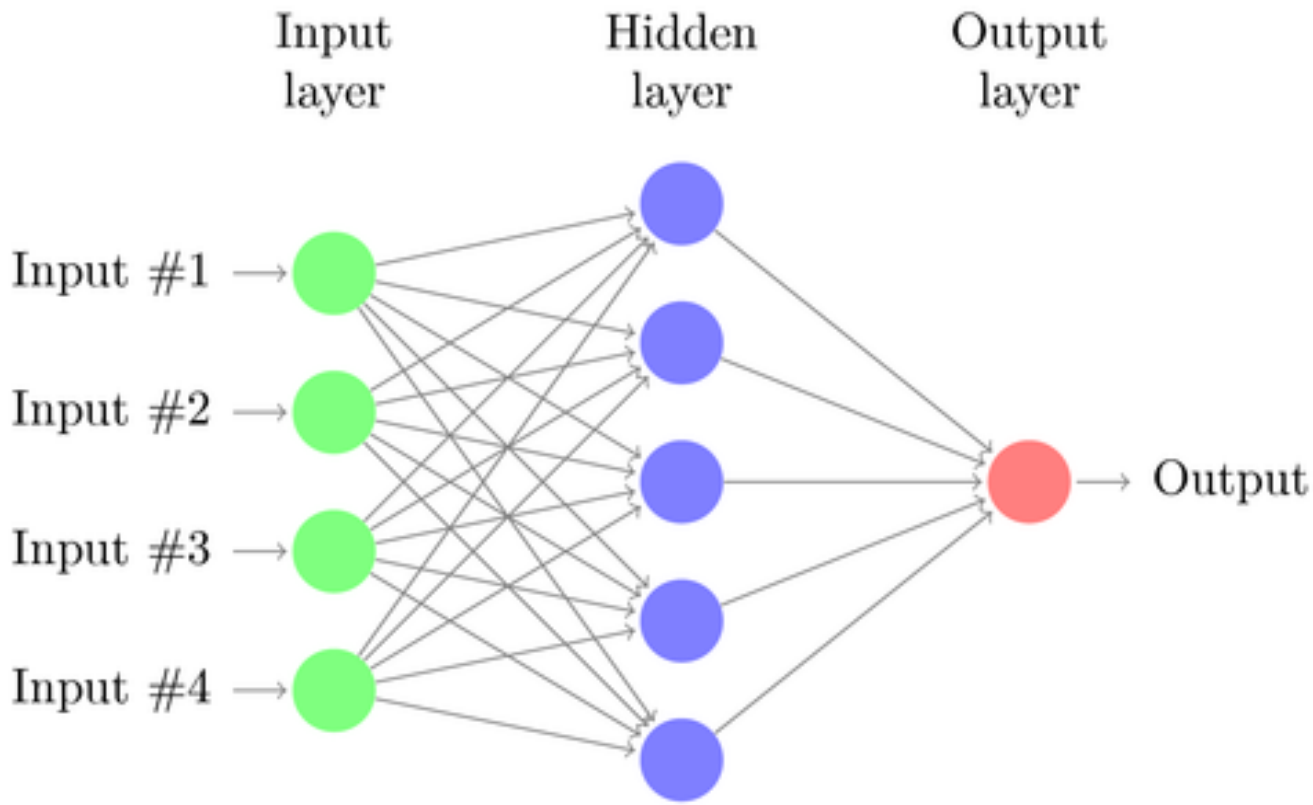
Control using Value Function Approximation

- 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
- Can be unstable. Generally involves intersection of the following:
  - Function approximation
  - Bootstrapping
  - **Off-policy learning**

RL with Function Approximation

- 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

Neural Networks [1]



[1] Figure by Kjell Magne Fauske

Deep Neural Networks (DNN)

- Composition of multiple functions
- Can use the chain rule to backpropagate the gradient
- Major innovation: tools to automatically compute gradients for a DNN

Deep Neural Networks (DNN) Specification and Fitting

- Generally combines both linear and non-linear transformations
  - Linear:
  - Non-linear:
- To fit the parameters, require a loss function (MSE, log likelihood etc)

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: 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
  - Can learn the parameters using stochastic gradient descent

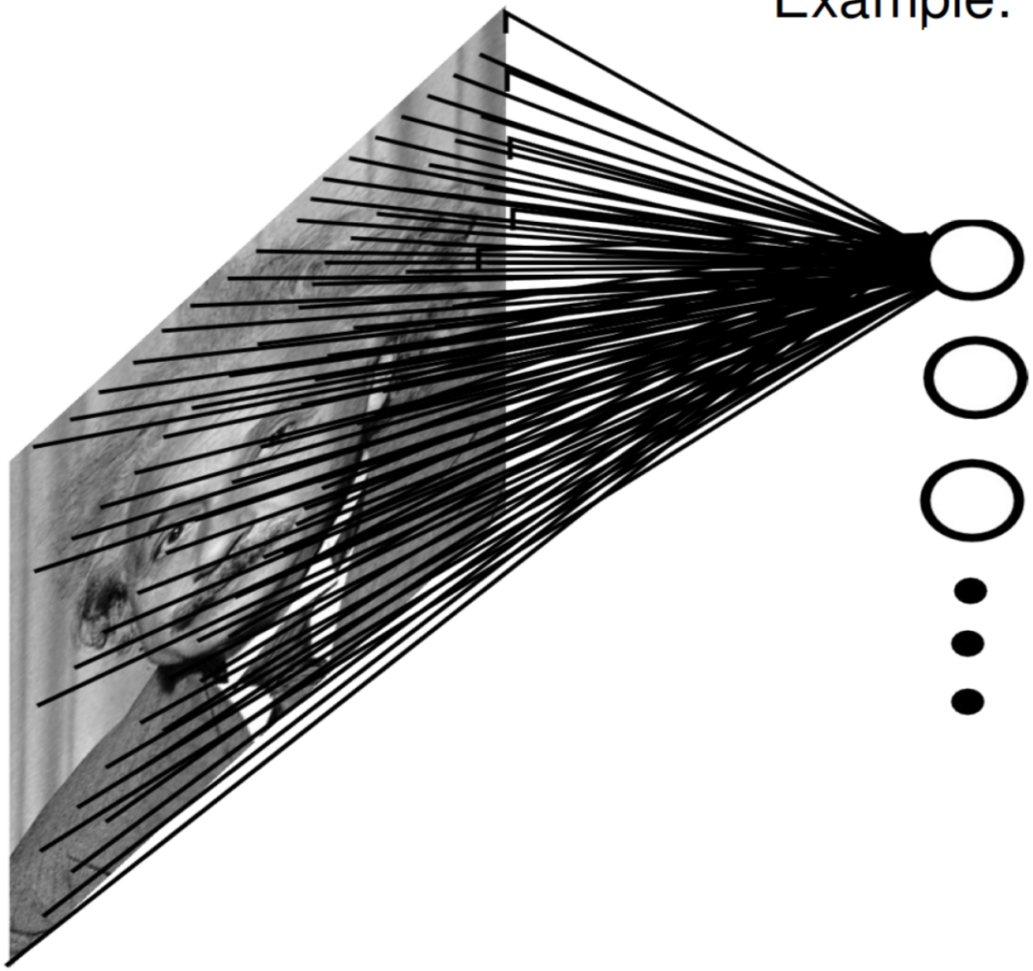
Today

- Value function approximation
- Deep neural networks
- **CNNs**
- DQN

Why Do We Care About CNNs?

- CNNs extensively used in computer vision
- If we want to go from pixels to decisions, likely useful to leverage insights for visual input

Fully Connected Neural Net

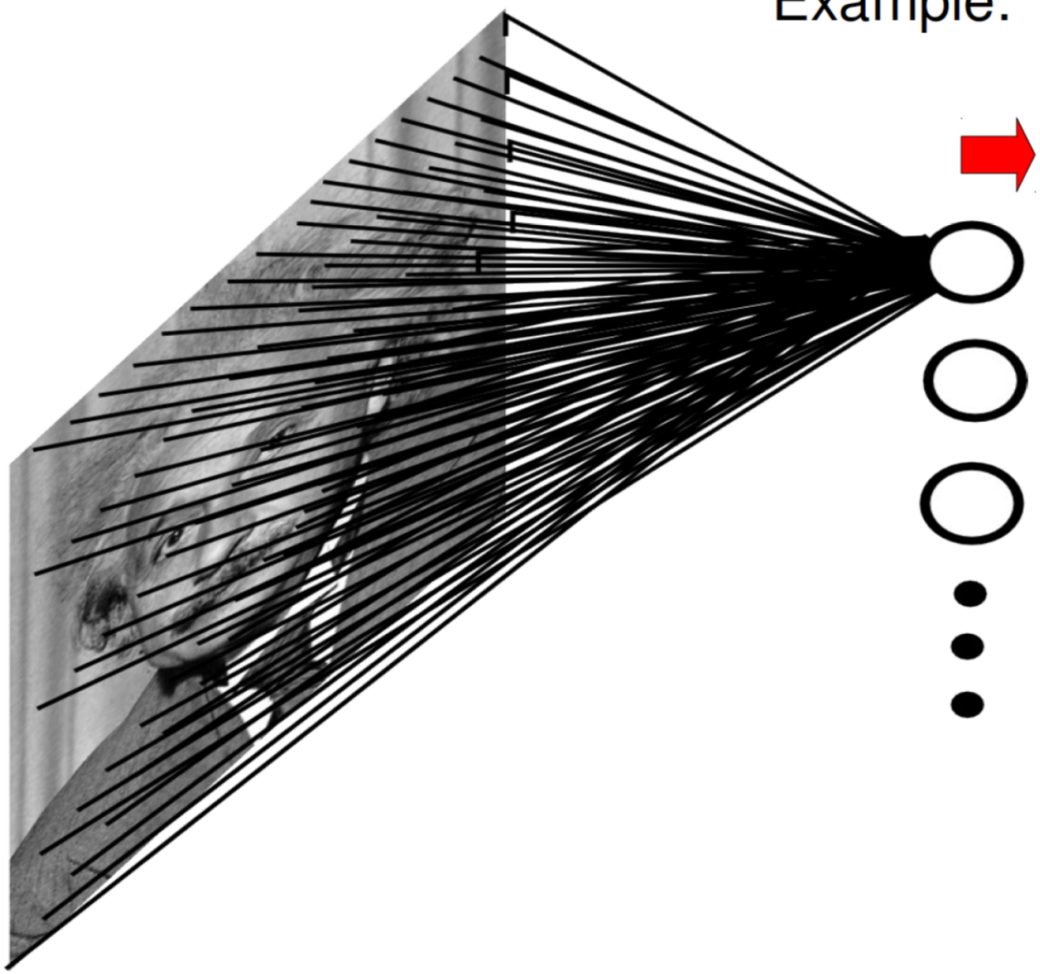


Example: 1000x1000 image

How many weight parameters for a single node which is a linear combination of input?

Ranzato

Fully Connected Neural Net



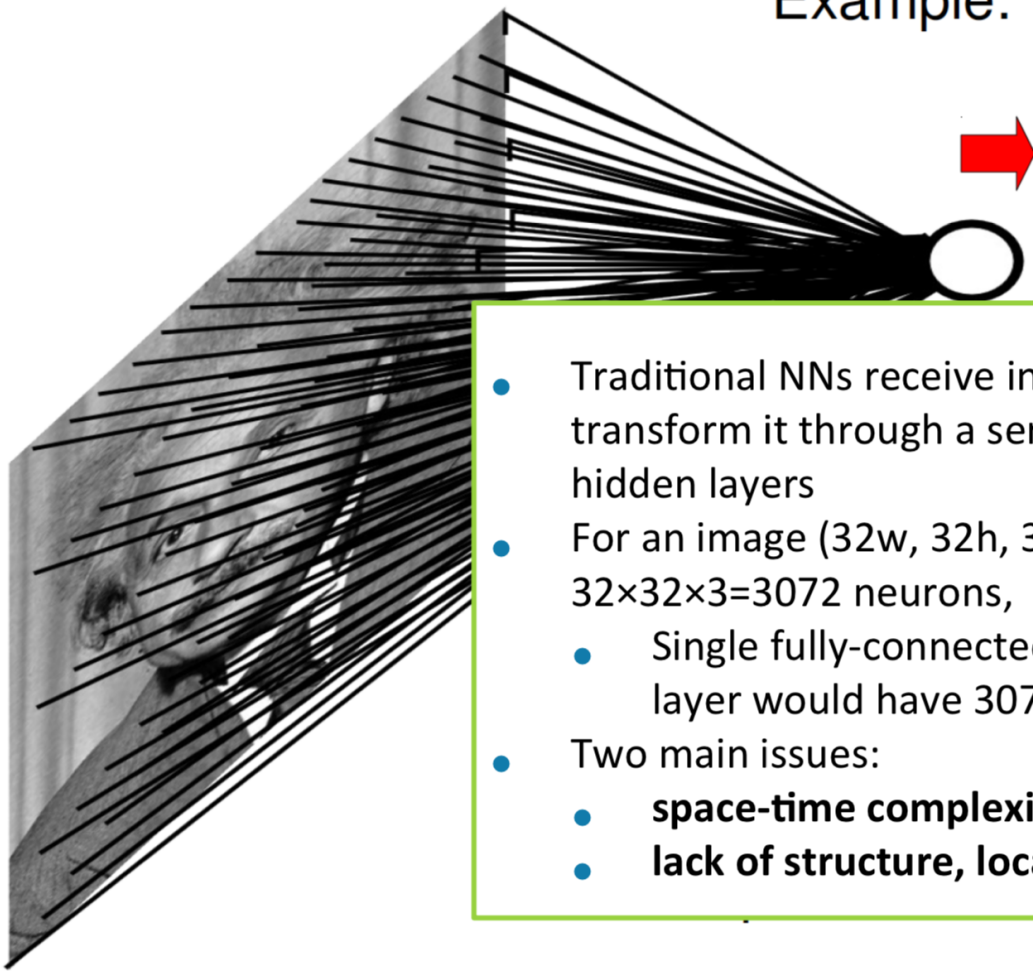
Example: 1000x1000 image

1M hidden units

10^12 parameters!!!

Ranzato

Fully Connected Neural Net



Example: 1000x1000 image
1M hidden units
10^12 parameters!!!

- Traditional NNs receive input as single vector & transform it through a series of (fully connected) hidden layers
- For an image (32w, 32h, 3c), the input layer has 32×32×3=3072 neurons,
  - Single fully-connected neuron in the first hidden layer would have 3072 weights …
- Two main issues:
  - **space-time complexity**
  - **lack of structure, locality of info**

Ranzato

Images Have Structure

- Have local structure and correlation
- Have distinctive features in space & frequency domains

Convolutional NN

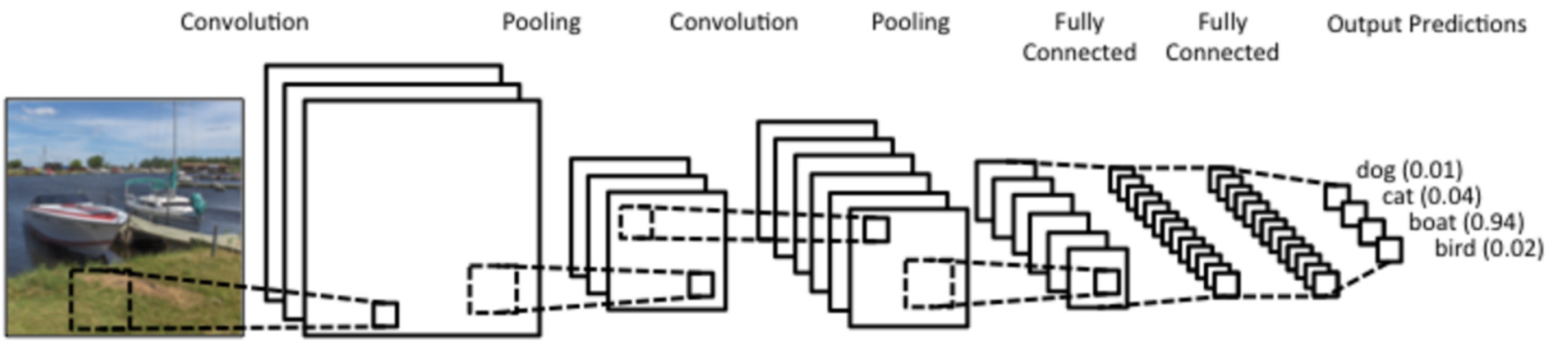
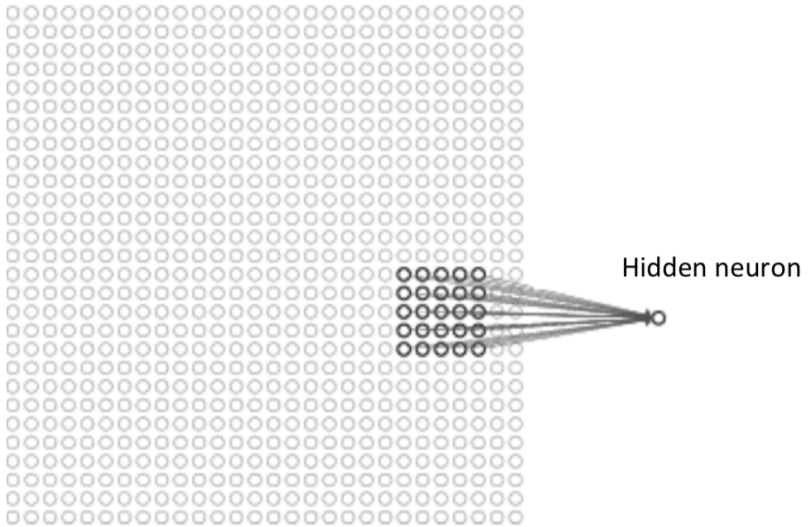
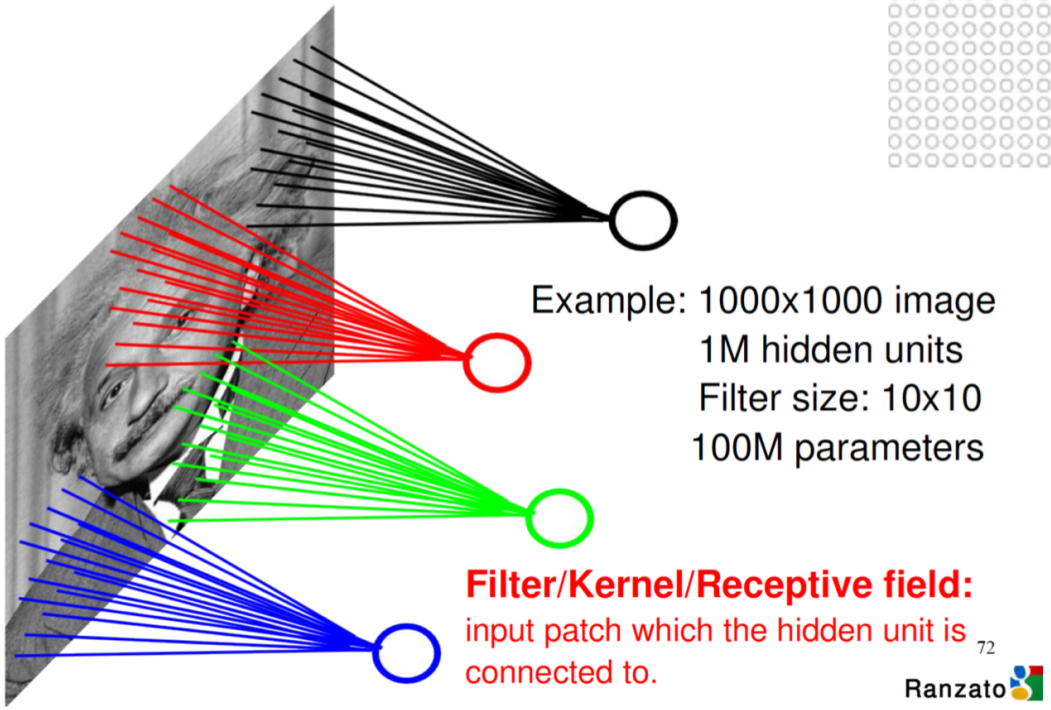


Image: http://d3kbpzbmcynnmx.cloudfront.net/wp-content/uploads/2015/11/Screen-Shot-2015-11-07-at-7.26.20-AM.png

- Consider local structure and common extraction of features
- Not fully connected
- Locality of processing
- Weight sharing for parameter reduction
- Learn the parameters of multiple convolutional filter banks
- Compress to extract salient features & favor generalization

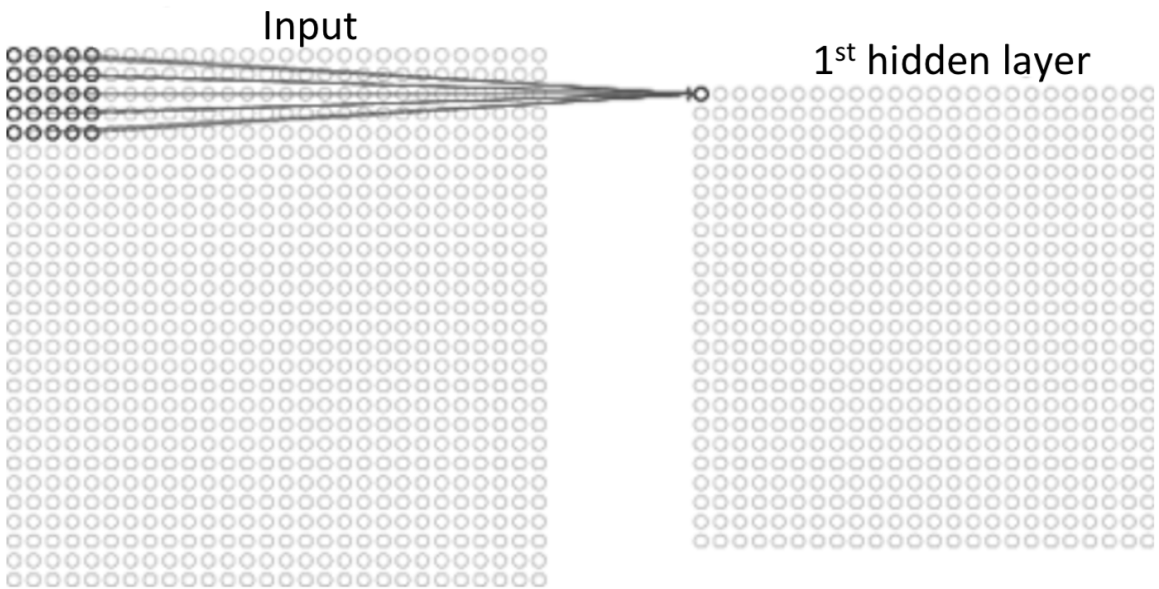
Locality of Information: Receptive Fields





(Filter) Stride

- Slide the 5x5 mask over all the input pixels
- Stride length $= 1$
  - Can use other stride lengths
- Assume input is 28x28, how many neurons in 1st hidden layer?



- Zero padding: how many 0s to add to either side of input layer

Shared Weights

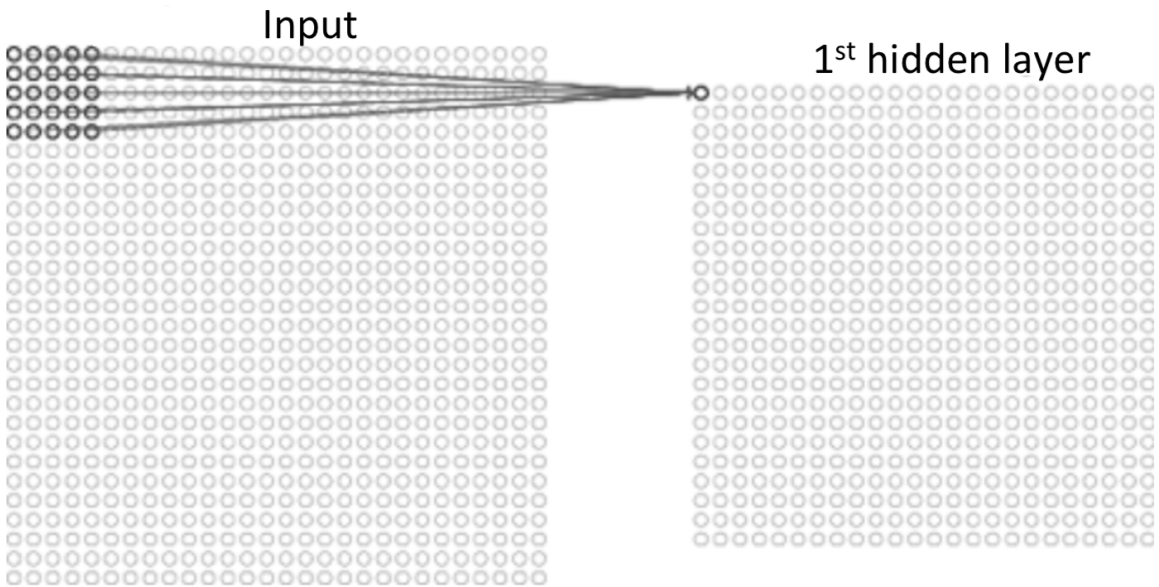
- What is the precise relationship between the neurons in the receptive field and that in the hidden layer?
- What is the *activation value* of the hidden layer neuron?

$$g(b + \sum_i w_i x_i)$$

- Sum over i is *only over the neurons in the receptive field* of the hidden layer neuron
- *The same weights w and bias b* are used for each of the hidden neurons
  - In this example, 24×24 hidden neurons

Ex. Shared Weights, Restricted Field

- Consider 28x28 input image
- 24x24 hidden layer

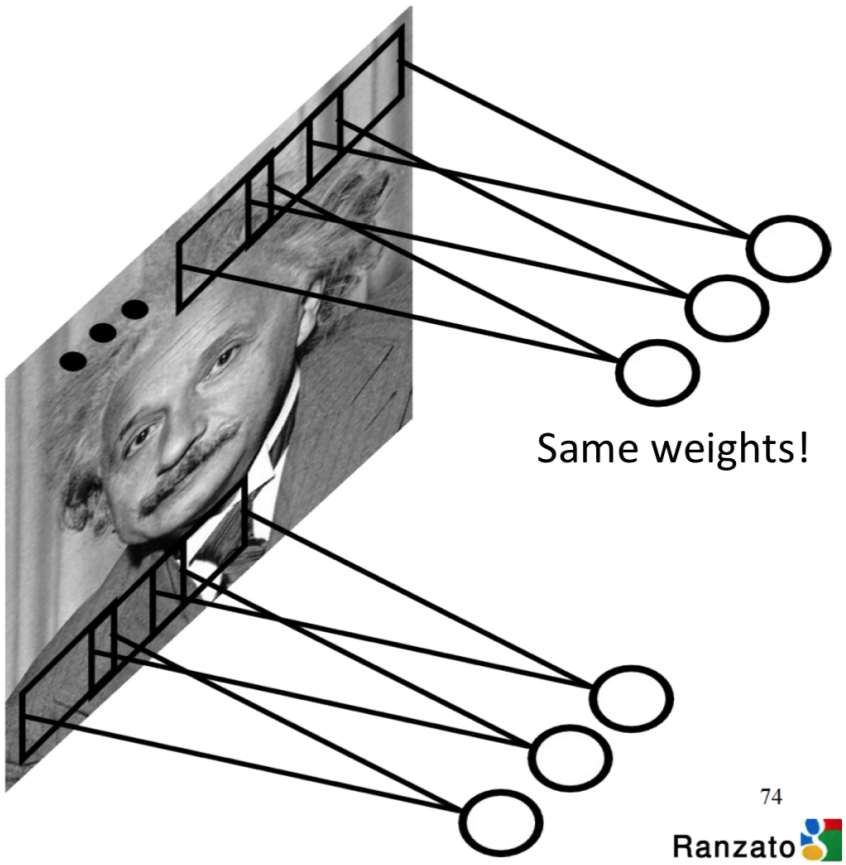


- Receptive field is 5x5

Feature Map

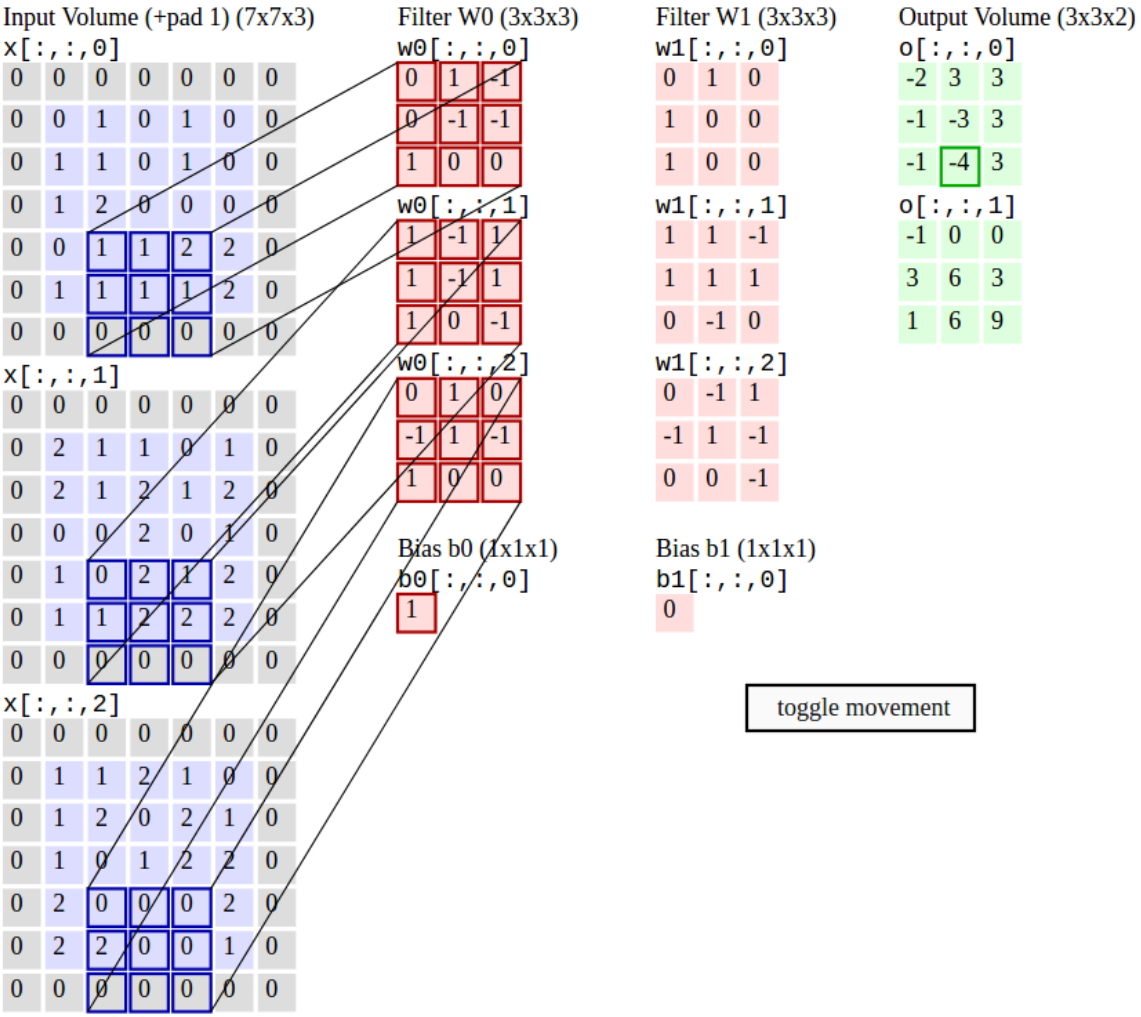
- All the neurons in the first hidden layer *detect exactly the same feature, just at different locations* in the input image.
- **Feature**: the kind of input pattern (e.g., a local edge) that makes the neuron produce a certain response level
- Why does this makes sense?
  - Suppose the weights and bias are (learned) such that the hidden neuron can pick out, a vertical edge in a particular local receptive field.
  - That ability is also likely to be useful at other places in the image.
  - Useful to apply the same feature detector everywhere in the image. Yields translation (spatial) invariance (try to detect feature at any part of the image)
  - Inspired by visual system

Feature Map



- The map from the input layer to the hidden layer is therefore a feature map: all nodes detect the same feature in different parts
- The map is defined by the shared weights and bias
- The shared map is the result of the application of a convolutional filter (defined by weights and bias), also known as convolution with learned kernels

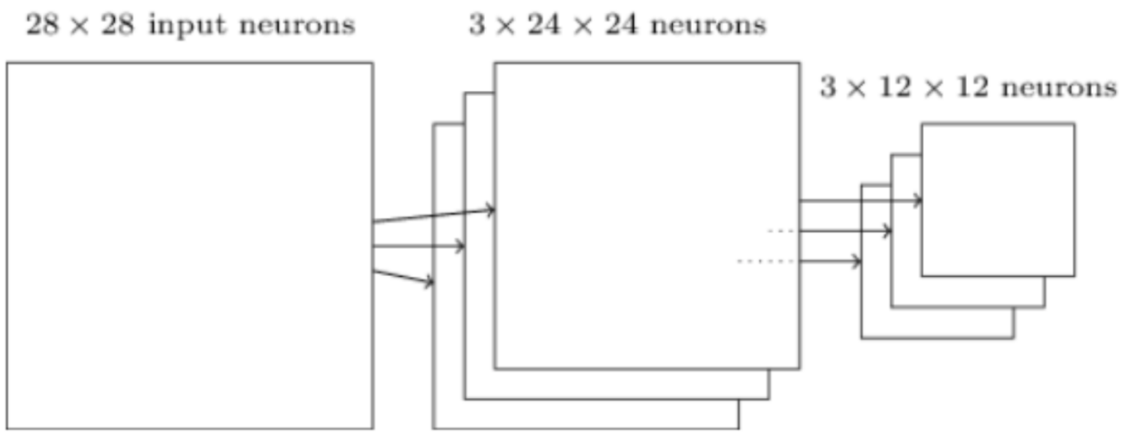
Convolutional Layer: Multiple Filters Ex.[2]



[2] http://cs231n.github.io/convolutional-networks/

Pooling Layers

- Pooling layers are usually used immediately after convolutional layers.
- Pooling layers simplify / subsample / compress the information in the output from convolutional layer
- A pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map



Final Layer Typically Fully Connected

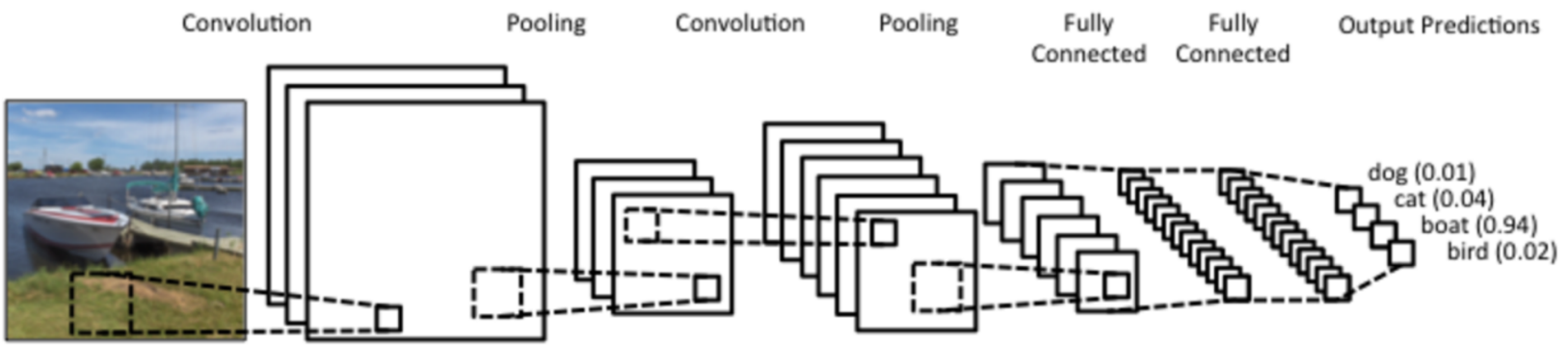


Image: http://d3kbpzbmcynnmx.cloudfront.net/wp-content/uploads/2015/11/Screen-Shot-2015-11-07-at-7.26.20-AM.png

Today

- Value function approximation
- Deep neural networks
- CNNs
- **DQN**

Generalization

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

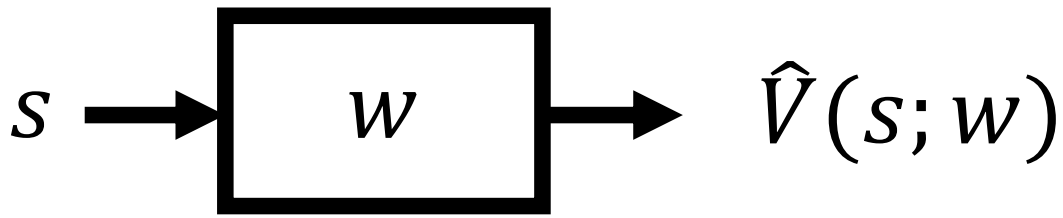
Deep Reinforcement Learning

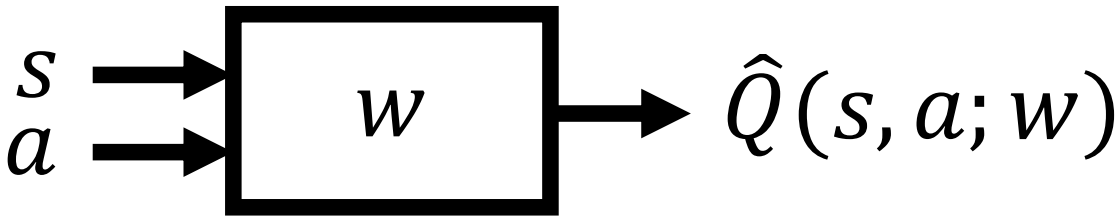
- Use deep neural networks to represent
  - Value, Q function
  - Policy
  - Model
- Optimize loss function by stochastic gradient descent (SGD)

Deep Q-Networks (DQNs)

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

$$\hat{Q}(s, a; \boldsymbol{w}) \approx Q(s, a)$$





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

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

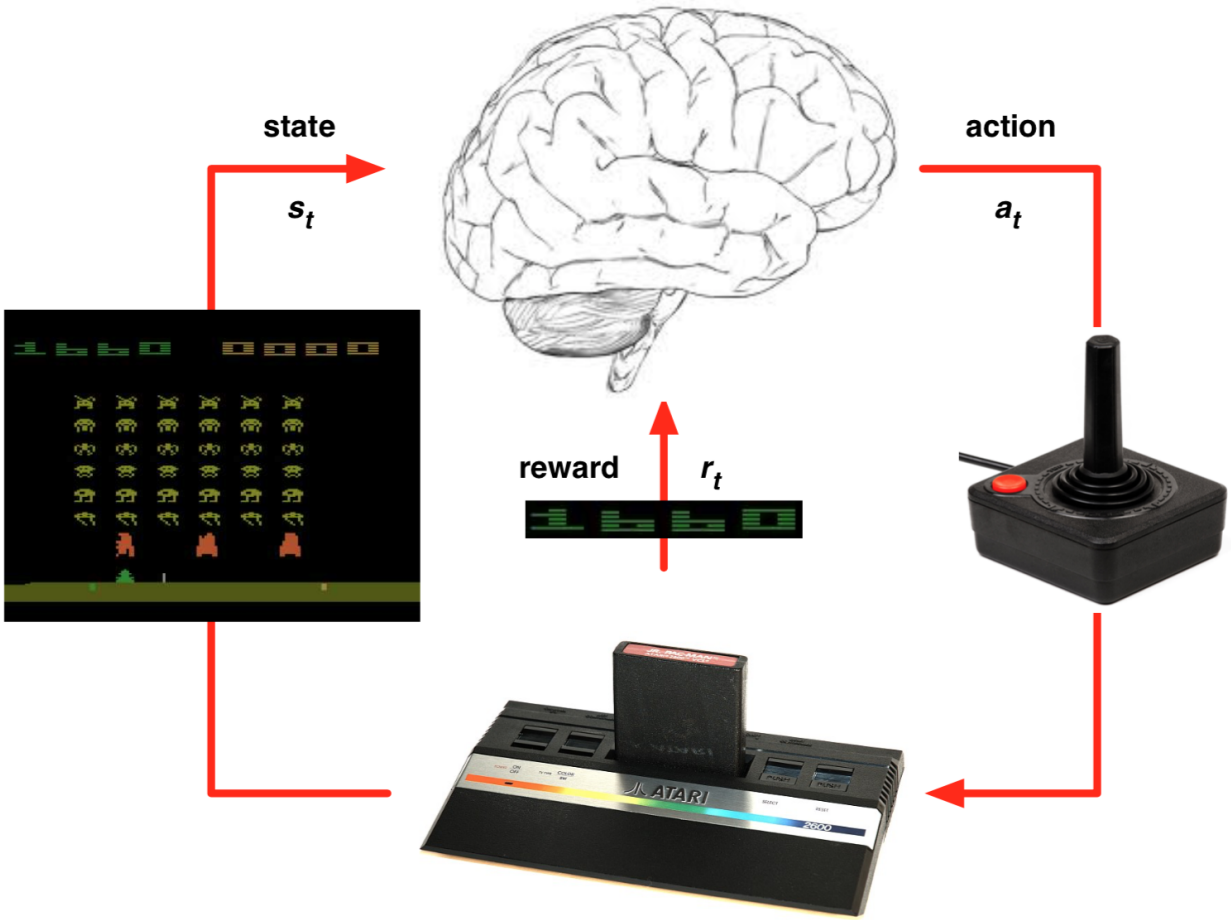
- For SARSA instead use a TD target $r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \boldsymbol{w})$ which leverages the current function approximation value

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \boldsymbol{w}) - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$$

- For Q-learning instead use a TD target $r + \gamma \max_a \hat{Q}(s_{t+1}, a; \boldsymbol{w})$ which leverages the max of the current function approximation value

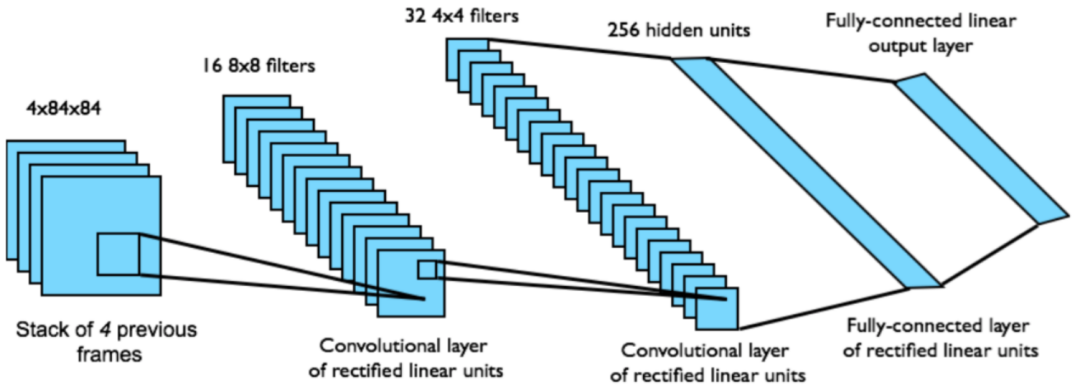
$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_a \hat{Q}(s_{t+1}, a; \boldsymbol{w}) - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$$

Using these ideas to do Deep RL in Atari



DQNs in Atari

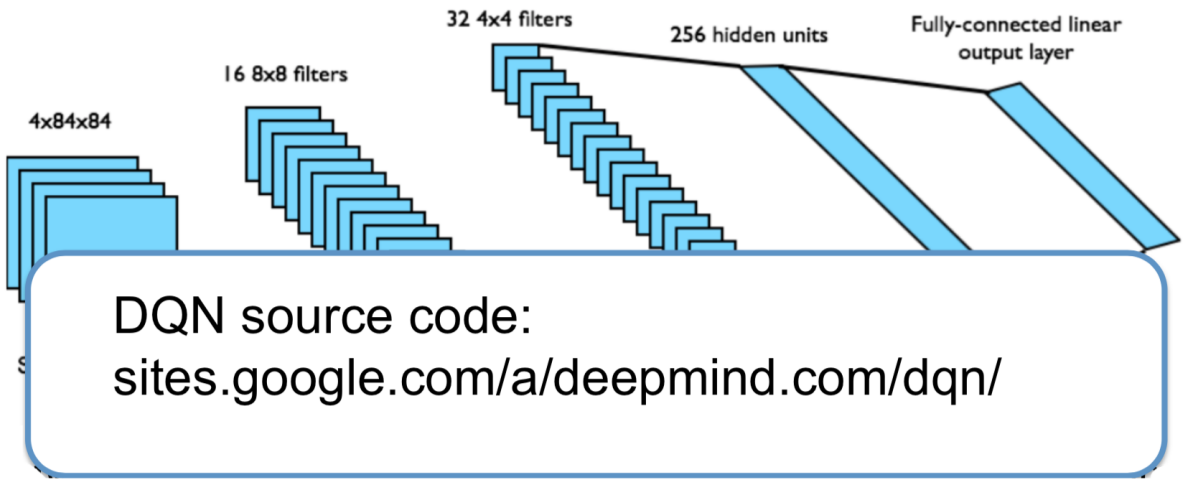
- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



- 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



- Network architecture and hyperparameters fixed across all games

Q-Learning with Value Function Approximation

- Q-learning converges to the optimal $Q^*(s, a)$ using table lookup representation
- In value function approximation Q-learning we can minimize MSE loss by stochastic gradient descent using a target Q estimate instead of true Q (as we saw with linear VFA)
- But Q-learning with VFA can diverge
- Two of the issues causing problems:
  - Correlations between samples
  - Non-stationary targets
- Deep Q-learning (DQN) addresses 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

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

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: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$
  - Use stochastic gradient descent to update the network weights

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

- **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 weights** used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights 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

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

DQN Pseudocode

Input C, α, $D = \{\}$, 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

DQN Pseudocode Hyperparameters

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

Note there are several hyperparameters and algorithm choices. One needs to choose the neural network architecture, the learning rate, and how often to update the target network. Often a fixed size replay buffer is used for experience replay, which introduces a parameter to control the size, and the need to decide how to populate it.

Check Your Understanding: Fixed Targets

- In DQN we compute the target value for the sampled (s, a, r, s) using a separate set of target weights: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-)$
- Select all that are true
- If the target network is trained on other data, this might help with the maximization bias
- This doubles the computation time compared to a method that does not have a separate set of weights
- This doubles the memory requirements compared to a method that does not have a separate set of weights
- Not sure

Check Your Understanding: Fixed Targets **Solutions**

- In DQN we compute the target value for the sampled (s, a, r, s') using a separate set of target weights: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-)$
- Select all that are true
- If the target network is trained on other data, this might help with the maximization bias
- This doubles the computation time compared to a method that does not have a separate set of weights
- This doubles the memory requirements compared to a method that does not have a separate set of weights
- Not sure

Answer: It doubles the memory requirements. In the maximization bias, we use a separate function to choose the action and to evaluate the value of it.

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

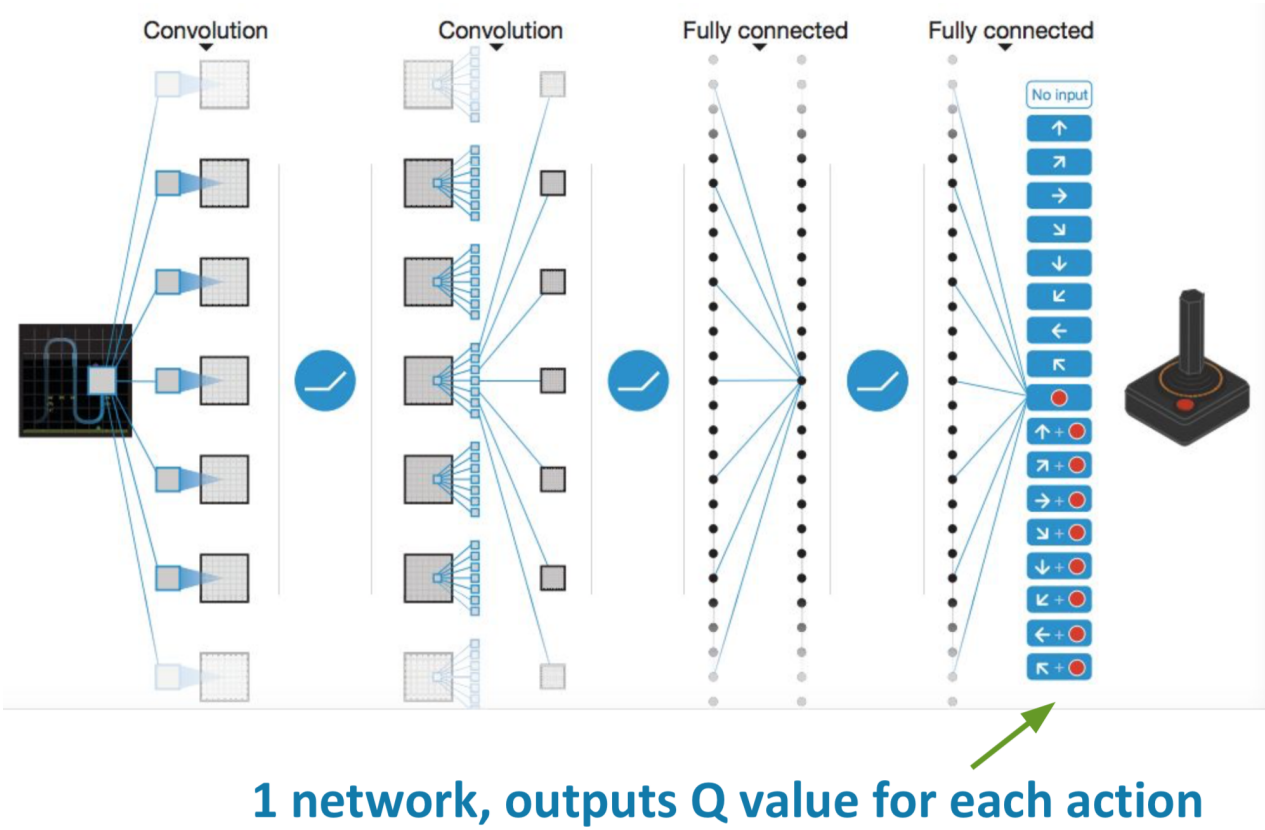


Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

DQN Results in Atari

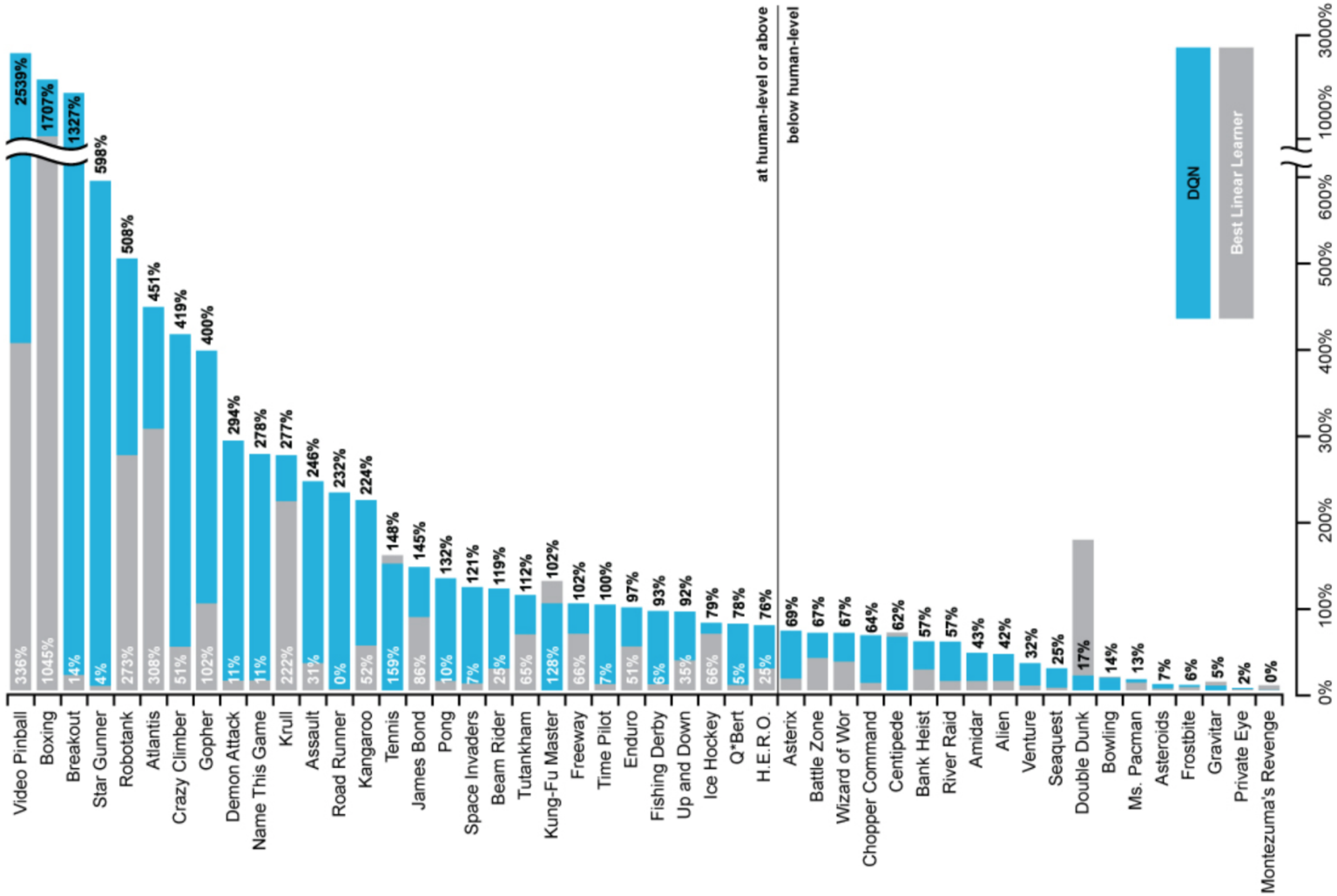


Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network
Breakout	3	3
Enduro	62	29
River Raid	2345	1453
Seaquest	656	275
Space Invaders	301	302

Note: just using a deep NN actually hurt performance sometimes!

Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network	DQN w/ fixed Q
Breakout	3	3	10
Enduro	62	29	141
River Raid	2345	1453	2868
Seaquest	656	275	1003
Space Invaders	301	302	373

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
- Why? Beyond helping with correlation between samples, what does replaying do?

Deep RL

- Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL
- Some immediate improvements (many others!)
  - **Double DQN** (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
  - Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
  - Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)

Class Structure

- Last time and start of this time: Control (making decisions) without a model of how the world works
- Rest of today: Deep reinforcement learning
- Next time: Deep RL continued

Lecture 7: Deep RL Continued

Emma Brunskill

CS234 Reinforcement Learning.

Refresh Your Knowledge 6

- Experience replay in deep Q-learning (select all):
  1. Involves using a bank of prior (s,a,r,s') tuples and doing Q-learning updates on the tuples in the bank
  2. Always uses the most recent history of tuples
  3. Reduces the data efficiency of DQN
  4. Increases the computational cost
  5. Not sure

Refresh Your Knowledge 6 Solutions

- Experience replay in deep Q-learning (select all):
  1. Involves using a bank of prior (s,a,r,s') tuples and doing Q-learning updates on the tuples in the bank
  2. Always uses the most recent history of tuples
  3. Reduces the data efficiency of DQN
  4. Increases the computational cost
  5. Not sure

Answer: It increases the computational cost, it uses a bank of tuples and it samples them, it's likely to **improve** the data efficiency, and it does not have to always use the most recent history of tuples.

Class Structure

- Last time: CNNs and Deep Reinforcement learning
- This time: DRL
- Next time: Policy Search

Deep RL

- Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL
- Some immediate improvements (many others!)
  - Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
  - Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
  - Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)

Today

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
- Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)
- Practical Tips

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

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**

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

  else

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

  end if
  $t = t + 1$
end loop

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

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

Double DQN

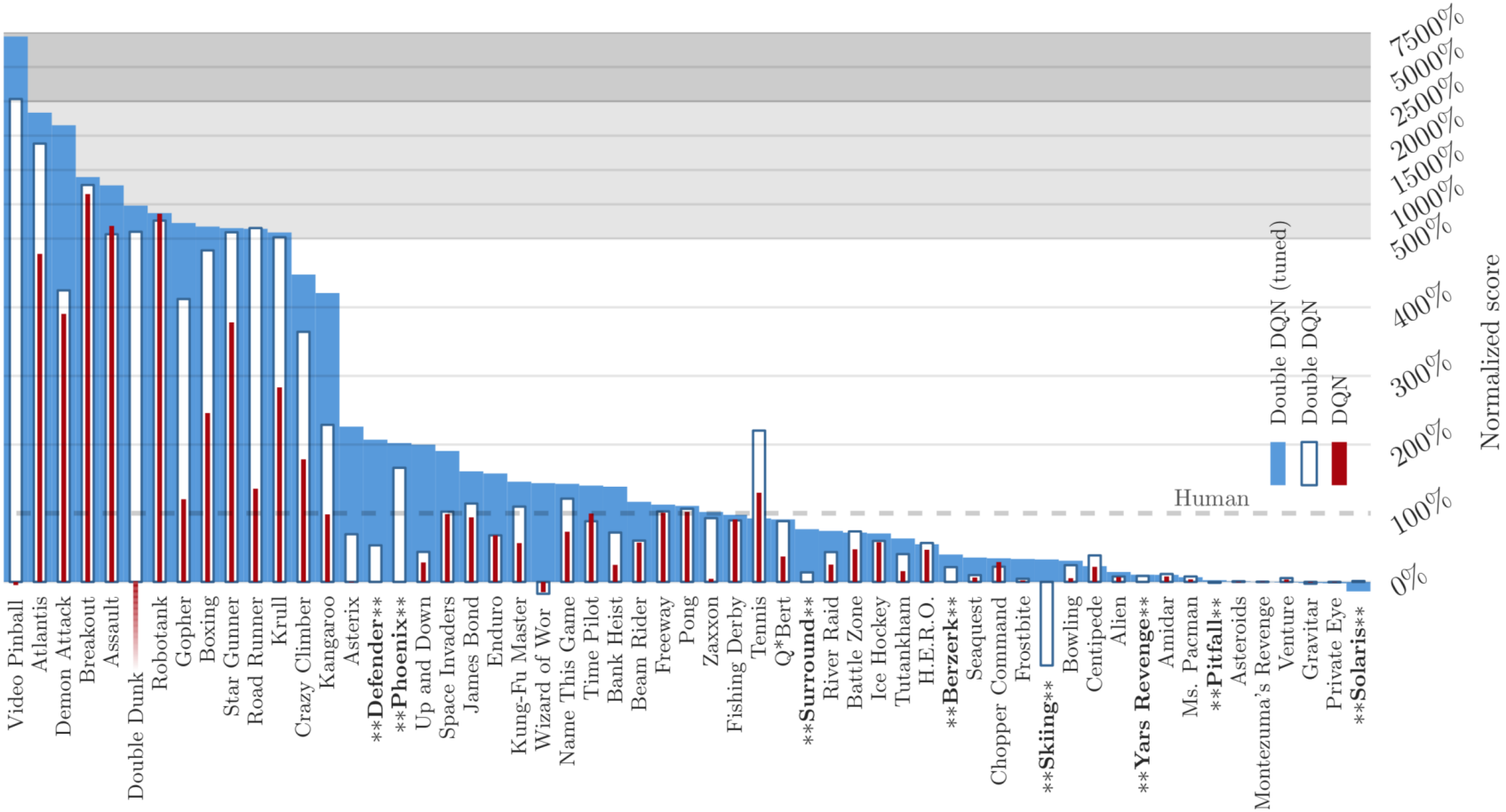


Figure: van Hasselt, Guez, Silver, 2015

Today

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
- Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)
- Practical Tips

Check Your Understanding: Mars Rover Model-Free Policy Evaluation

s_1	s_2	s_3	s_4	s_5	s_6	s_7
$R(s_1) = +1$ *Okay Field Site*	$R(s_2) = 0$	$R(s_3) = 0$	$R(s_4) = 0$	$R(s_5) = 0$	$R(s_6) = 0$	$R(s_7) = +10$ *Fantastic Field Site*

- $\pi(s) = a_1 \ \forall s$, $\gamma = 1$. Any action from s_1 and s_7 terminates episode
- Trajectory = (s_3, a_1, 0, s_2, a_1, 0, s_2, a_1, 0, s_1, a_1, 1, terminal)
- First visit MC estimate of V of each state? [1 1 1 0 0 0 0]
- TD estimate of all states (init at 0) with $\alpha = 1$ is [1 0 0 0 0 0 0]
- Choose 2 additional "replay" backups to do. Which should we pick to get a V estimate closest to MC first visit estimate?
  1. Doesn't matter, any will yield the same
  2. $(s_3, a_1, 0, s_2)$ then $(s_2, a_1, 0, s_1)$
  3. $(s_2, a_1, 0, s_1)$ then $(s_2, a_1, 0, s_2)$
  4. $(s_2, a_1, 0, s_1)$ then $(s_3, a_1, 0, s_2)$
  5. Not sure

Check Your Understanding: Mars Rover Model-Free Policy Evaluation Solution

s_1	s_2	s_3	s_4	s_5	s_6	s_7
$R(s_1) = +1$ *Okay Field Site*	$R(s_2) = 0$	$R(s_3) = 0$	$R(s_4) = 0$	$R(s_5) = 0$	$R(s_6) = 0$	$R(s_7) = +10$ *Fantastic Field Site*

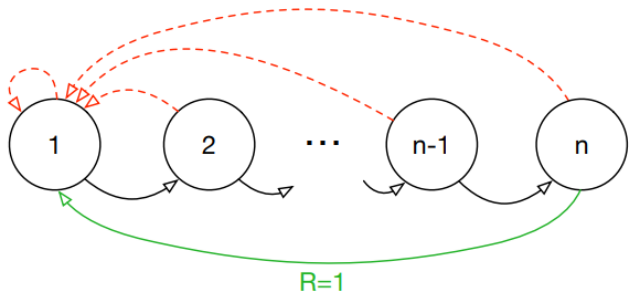
- $\pi(s) = a_1 \ \forall s, \ \gamma = 1$. Any action from s_1 and s_7 terminates episode
- Trajectory = (s_3, a_1, 0, s_2, a_1, 0, s_2, a_1, 0, s_1, a_1, 1, terminal)
- First visit MC estimate of V of each state? [1 1 1 0 0 0 0]
- TD estimate of all states (init at 0) with $\alpha = 1$ is [1 0 0 0 0 0 0]
- Choose 2 additional "replay" backups to do. Which should we pick to get a V estimate closest to MC first visit estimate?
  1. Doesn't matter, any will yield the same
  2. $(s_3, a_1, 0, s_2)$ then $(s_2, a_1, 0, s_1)$
  3. $(s_2, a_1, 0, s_1)$ then $(s_2, a_1, 0, s_2)$
  4. $(s_2, a_1, 0, s_1)$ then $(s_3, a_1, 0, s_2)$
  5. Not sure

  Answer: $(s_2, a_1, 0, s_1)$, $(s_3, a_1, 0, s_2)$ yielding V =[1 1 1 0 0 0 0].

Impact of Replay?

- In tabular TD-learning, **order** of replaying updates could help speed learning
- Repeating some updates seems to better propagate info than others
- Systematic ways to prioritize updates?

Potential Impact of Ordering Episodic Replay Updates



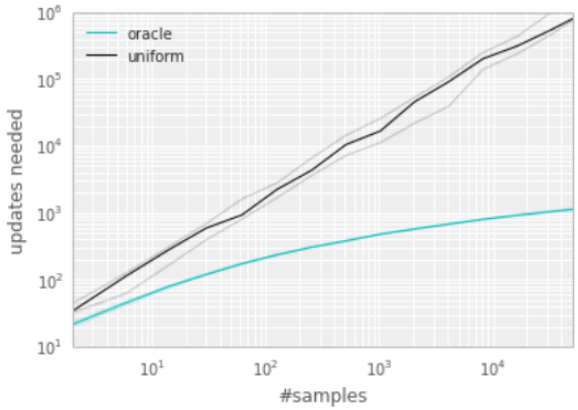


Figure: Schaul, Quan, Antonoglou, Silver ICLR 2016

- Schaul, Quan, Antonoglou, Silver ICLR 2016
- Oracle: picks (s, a, r, s') tuple to replay that will minimize global loss
- Exponential improvement in convergence
  - Number of updates needed to converge
- Oracle is not a practical method but illustrates impact of ordering

Prioritized Experience Replay

- Let i be the index of the i-th tuple of experience (s_i, a_i, r_i, s_{i+1})
- Sample tuples for update using priority function
- Priority of a tuple i is proportional to DQN error

$$p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a'; \boldsymbol{w}^-) - Q(s_i, a_i; \boldsymbol{w}) \right|$$

- Update p_i every update. p_i for new tuples is set to maximum value
- One method[1]: proportional (stochastic prioritization)

$$P(i) = \frac{p_i^\beta}{\sum_k p_k^\beta}$$

[1] See paper for details and an alternative

Check Your Understanding: Prioritized Replay

- Let i be the index of the i-th tuple of experience (s_i, a_i, r_i, s_{i+1})
- Sample tuples for update using priority function
- Priority of a tuple i is proportional to DQN error

$$p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a'; \boldsymbol{w}^-) - Q(s_i, a_i; \boldsymbol{w}) \right|$$

- Update p_i every update.
- One method [See paper for details]: proportional (stochastic prioritization)

$$P(i) = \frac{p_i^\beta}{\sum_k p_k^\beta}$$

- $\beta = 0$ yields what rule for selecting among existing tuples?
- Selects randomly
- Selects the one with the highest priority
- It depends on the priorities p of the tuples
- Not Sure

Check Your Understanding: Prioritized Replay

- Let i be the index of the i-th tuple of experience (s_i, a_i, r_i, s_{i+1})
- Sample tuples for update using priority function
- Priority of a tuple i is proportional to DQN error

$$p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a'; \boldsymbol{w}^-) - Q(s_i, a_i; \boldsymbol{w}) \right|$$

- Update p_i every update.
- One method[1]: proportional (stochastic prioritization)

$$P(i) = \frac{p_i^\beta}{\sum_k p_k^\beta}$$

- $\beta = 0$ yields what rule for selecting among existing tuples?
- Selects randomly
- Selects the one with the highest priority
- It depends on the priorities of the tuples
- Not Sure

Answer: Selects randomly

Performance of Prioritized Replay vs Double DQN

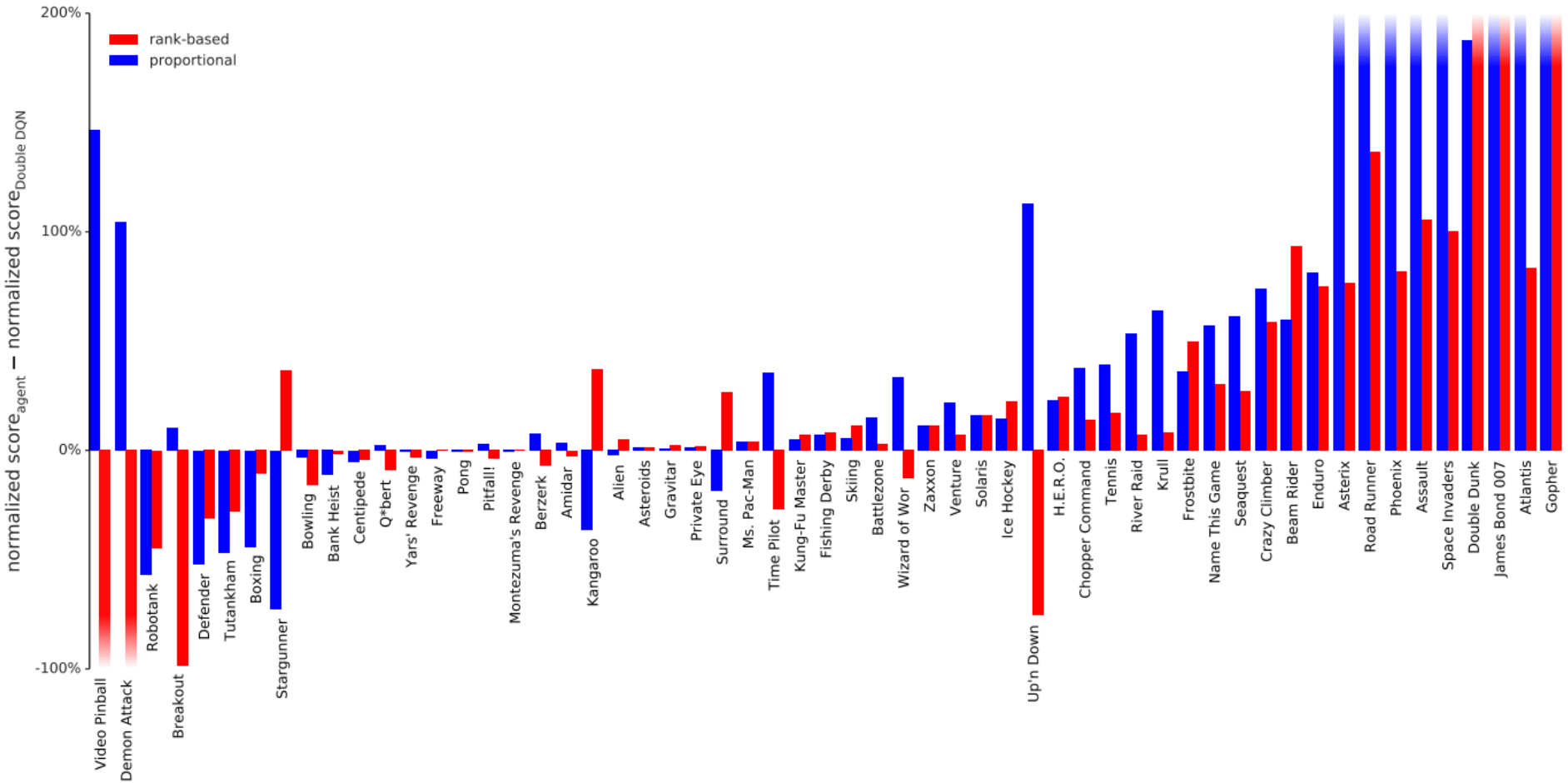


Figure: Schaul, Quan, Antonoglou, Silver ICLR 2016

Today

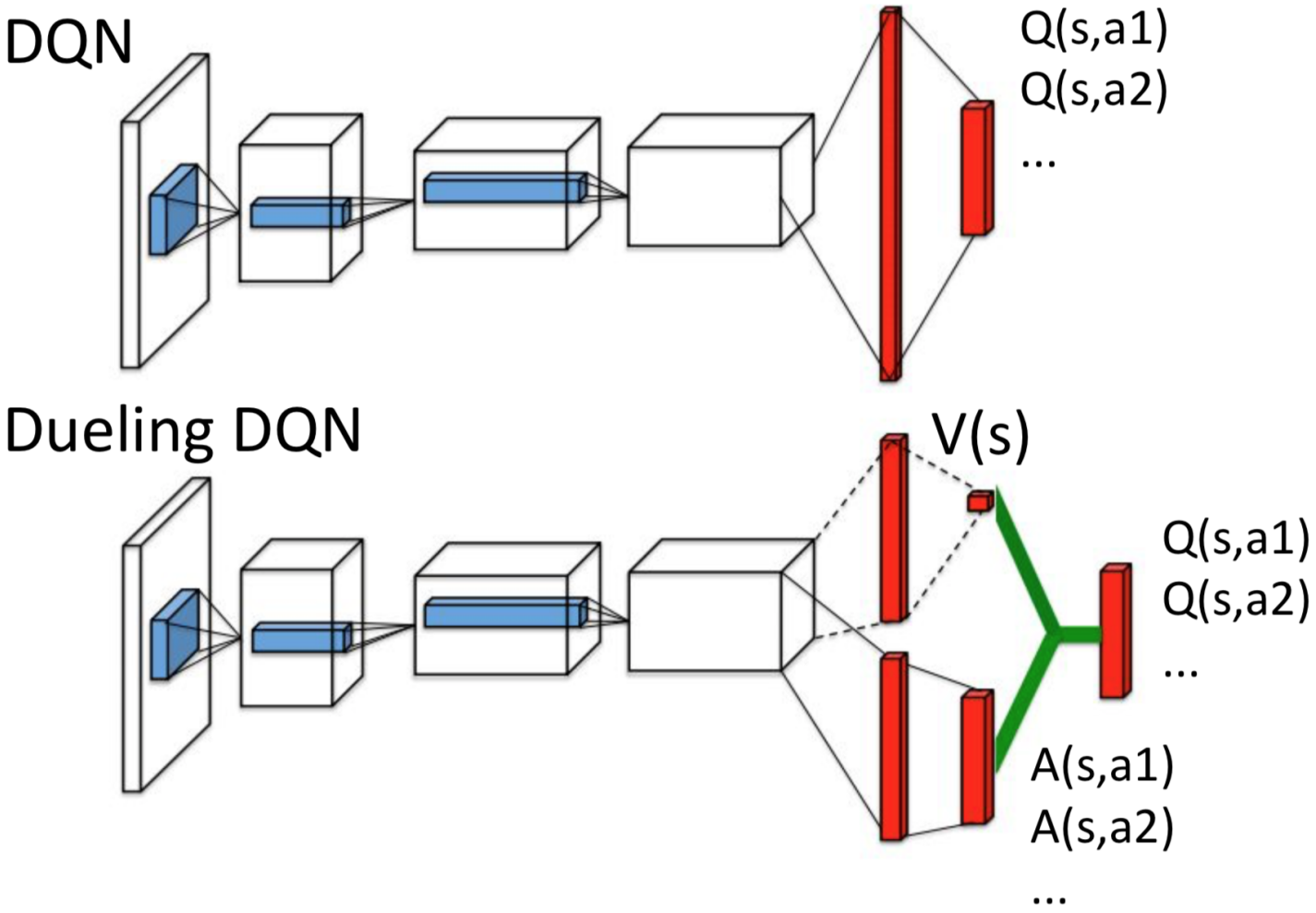
- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
- Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)
- Practical Tips

Value & Advantage Function

- Intuition: Features needed to accurately represent value may be different than those needed to specify difference in actions
- E.g.
  - Game score may help accurately predict $V(s)$
  - But not necessarily in indicating relative action values $Q(s, a_1)$ vs $Q(s, a_2)$
- Advantage function (Baird 1993)

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

Dueling DQN



Wang et.al., ICML, 2016

Advantage Function and Training

- Advantage function

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

- Consider a network that outputs $V(s;\theta,\beta)$ as well as advantage $A(s,a;\theta,\lambda)$ where θ,β, and λ are parameters
- To construct Q could use $Q(s,a;\theta,\beta,\lambda) = V(s;\theta,\beta) + A(s,a;\theta,\lambda)$
- Do we expect that this architecture will result in learning a good estimate of true V or A?

Check Your Understanding: Unique?

- Advantage function

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

- For a given Q function, is there a unique A advantage function and V?
  1. Yes
  2. No
  3. Not sure

Check Your Understanding: Unique?

- Advantage function

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

- For a given Q function, is there a unique A advantage function and V?
  1. Yes
  2. No
  3. Not sure

  Answer: No. If we are just given a Q, there are many A and V that could satisfy this – for example, by shifting things by a constant. This can cause challenges for using the simple proposal before: $Q(s,a;\theta,\beta,\lambda) = V(s;\theta,\beta) + A(s,a;\theta,\lambda)$

Uniqueness

- Consider a network that outputs $V(s;\theta,\beta)$ as well as advantage $A(s,a;\theta,\lambda)$ where θ,β, and λ are parameters
- To construct Q could use $Q(s,a;\theta,\beta,\lambda) = V(s;\theta,\beta) + A(s,a;\theta,\lambda)$
- Option 1: Force $Q(s,a) = V(s)$ for the best action suggested by the advantage:

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

- This helps force the V network to approximate V

Uniqueness

- Consider a network that outputs $V(s;\theta,\beta)$ as well as advantage $A(s,a;\theta,\lambda)$ where θ,β, and λ are parameters
- To construct Q could use $Q(s,a;\theta,\beta,\lambda) = V(s;\theta,\beta) + A(s,a;\theta,\lambda)$
- Option 1: Force $Q(s,a) = V(s)$ for the best action suggested by the advantage:

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

- This helps force the V network to approximate V
- Option 2: Use mean as baseline (more stable)

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

- More stable often because averaging over all advantages instead of the advantage of the current max action.

Dueling DQN V.S. Double DQN with Prioritized Replay

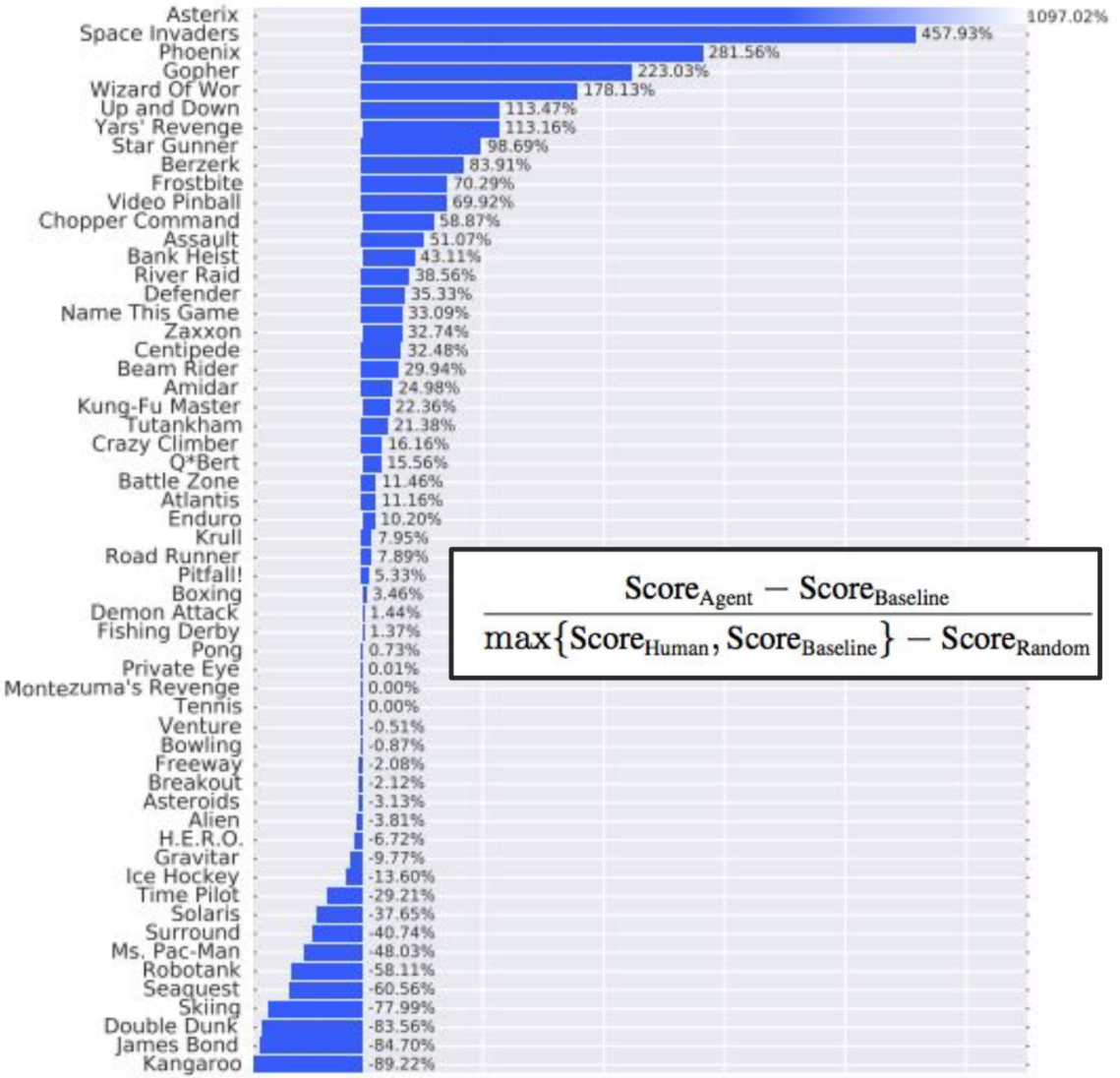
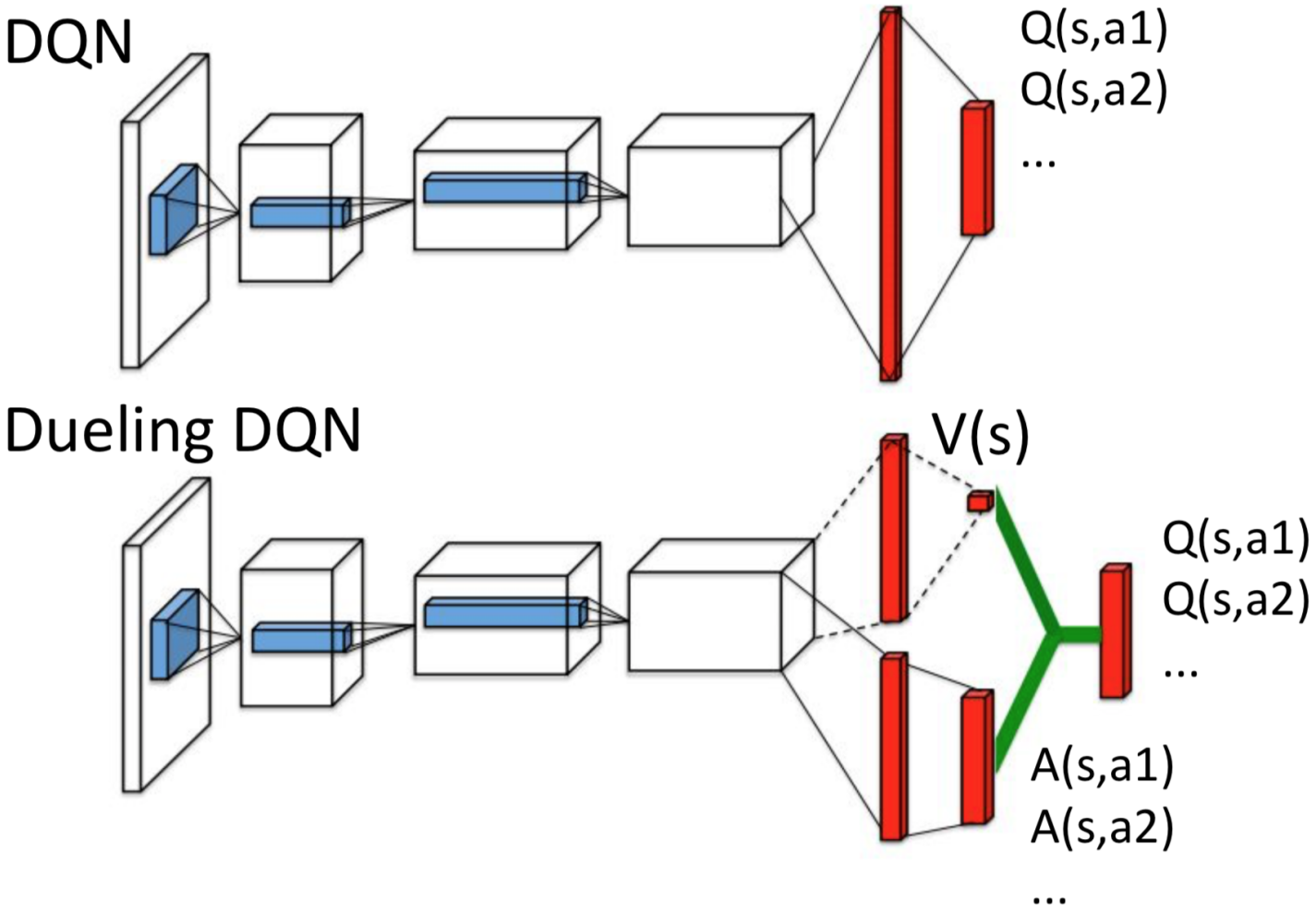


Figure: Wang et al, ICML 2016

Dueling DQN



Wang et.al., ICML, 2016

Today

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
- Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)
- **Practical Tips**

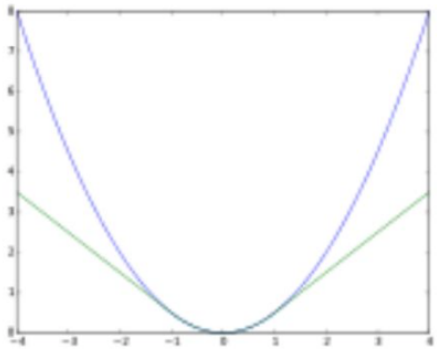
Practical Tips for DQN on Atari (from J. Schulman)

- 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

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

- Try Huber loss on Bellman error

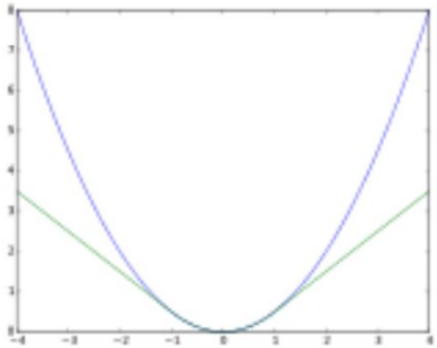
$$L(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \leq \delta \\ \delta|x| - \frac{\delta^2}{2} & \text{otherwise} \end{cases}$$



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

- Try Huber loss on Bellman error

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

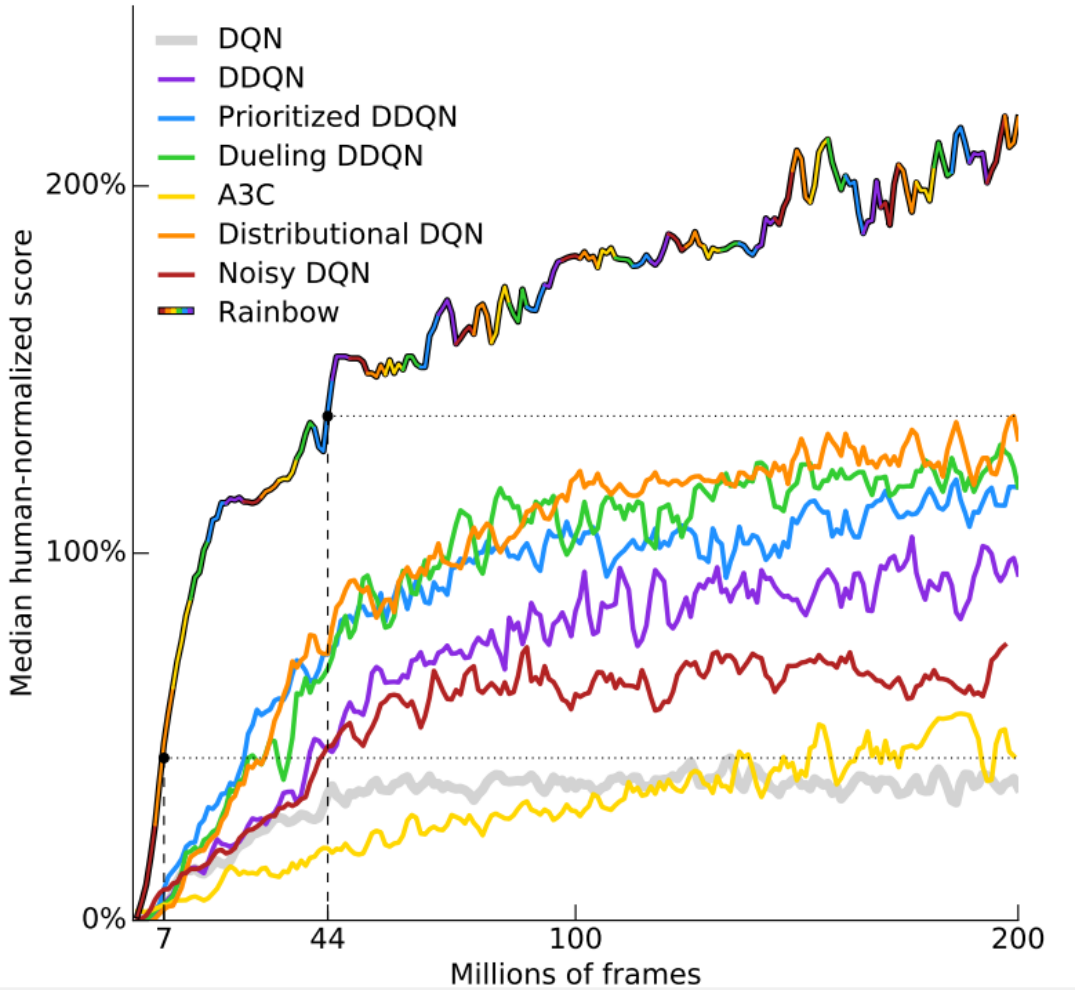


- Consider trying Double DQN—significant improvement from small code change
- To test out your data pre-processing, try your own skills at navigating the environment based on processed frames
- Always run at least two different seeds when experimenting
- Learning rate scheduling is beneficial. Try high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

Recap: Deep Model-free RL, 3 of the Early Big Ideas

- Double DQN: (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
- Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)

Deep Reinforcement Learning 2018



- Hessel, Matteo, et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning."
- Very active area of research!

Summary of Model Free Value Function Approximation with DNN & What You Should Know

- DNN are very expressive function approximators
- Can use DNNs to represent the Q function and do MC or TD style methods
- You should be able to implement DQN (assignment 2)
- You should be able to list a few extensions that help performance beyond DQN

Class Structure

- Last time: CNNs and Deep Reinforcement learning
- This time: Deep RL
- Next time: Policy Search