Lecture 5: Value Function Approximation

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2023

Draws from

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

L5 Refresh Your Knowledge

- 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

L5 Refresh Your Knowledge

- 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 (True)
  2. SARSA will converge to the optimal Q-values (False) (need GLIE)
  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

Table of Contents

A note on Monte Carlo vs TD estimates

hat just to emphasize estimate

- Policy evaluation: $\hat{V}^\pi \leftarrow (1-\alpha)\hat{V}^\pi + \alpha V_{target}$
- MC: $V_{target}(s_t) = G_t$ (sum of discounted returns until the episode terminates)
  - Target is unbiased estimate of V^π
  - Target can be high variance
- TD(0): $V_{target}(s_t) = r_t + \gamma \hat{V}(s')$
  - Target is a biased estimate of V^π
  - Target is lower variance
- Which one should we use? Is there other alternatives?

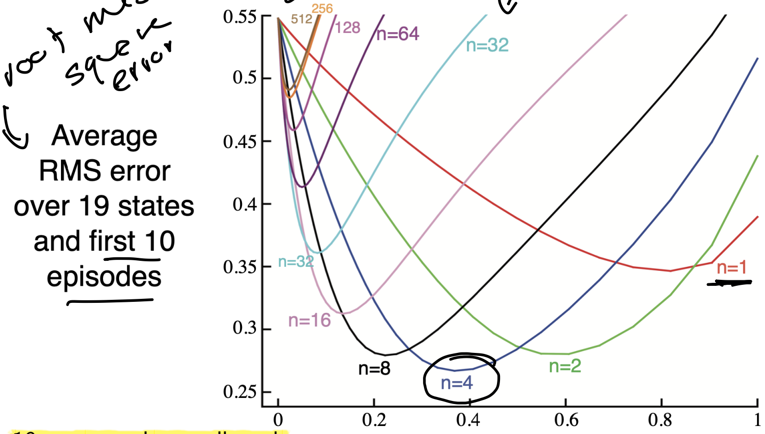
1 $V_{target} = 1 + 0$
$= 1$

n-step TD estimates

- Policy evaluation: $\hat{V}^{\pi} \leftarrow (1-\alpha)\hat{V}^{\pi} + \alpha V_{target}$
- MC: $V_{target}(s_t) = G_t$ (sum of discounted returns until the episode terminates)
  - Target is unbiased estimate of V^{π}
  - Target can be high variance
- TD(0): $V_{target}(s_t) = r_t + \gamma \hat{V}(s')$
  - Target is a biased estimate of V^{π}
  - Target is lower variance
- Best of both worlds?
- **n-step TD**: $V_{target}(s_t) = r_t + \gamma r_{t+1} + \gamma r_{t+2} + ... \gamma^n \hat{V}(s_{t+n})$ bootstraps

TD(λ) Sutton & Barto

1



- 19 state random walk task.

[1] Figure 7.2 from Sutton and Barto 2018

Table of Contents

Feature Vectors

- Use a feature vector to represent a state s

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

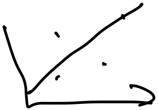
Recall: Linear Value Function Approximation for Prediction With An Oracle

- 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: $\Delta \boldsymbol{w} = -\frac{1}{2}\alpha(V^\pi(s) - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}$
- Update = step-size × prediction error × feature value

Table of Contents

Recall: 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

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

general

linear

linear

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

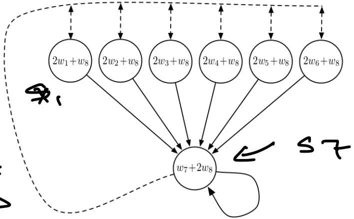
MC Linear Value Function Approximation for Policy Evaluation

```
Initialize w = 0, k = 1
loop
    Sample k-th episode (s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, ..., s_{k,L_k}) given π
    for t = 1, ..., L_k do
        if First visit to (s) in episode k then
            G_t(s) = Σ_{j=t}^{L_k} r_{k,j}        (this could include γ)
            Update weights: Δw = α(G_t − x(s_t)^T w) x(s_t)
                                   target ≈ V^π
        end if
    end for
    k = k + 1
end loop
```

$$G_t(s) = \sum_{j=t}^{L_k} r_{k,j}$$

$$\Delta \boldsymbol{w} = \alpha(\underbrace{G_t}_{\text{target} \approx V^\pi} - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t)$$

Baird (1995)-Like Example with MC Policy Evaluation[2]



- $\boldsymbol{x}(s_1) = [2\ 0\ 0\ 0\ 0\ 0\ 0\ 1]$ $\boldsymbol{x}(s_2) = [0\ 2\ 0\ 0\ 0\ 0\ 0\ 1]$ $\ldots \boldsymbol{x}(s_6) = [0\ 0\ 0\ 0\ 0\ 2\ 0\ 1]$ $\boldsymbol{x}(s_7) = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 2]$ $\quad r(s) = 0\ \forall s$ $\quad$ 2 actions a_1 solid line, a_2 dotted
- Small prob s_7 goes to terminal state s_T
- Consider trajectory $(s_1, a_1, 0, s_7, a_1, 0, s_7, a_1, 0, s_T)$. $G(s_1) = 0$ $\quad \gamma = 1$
- Let $w_0 = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$. MC update: $\Delta \boldsymbol{w} = \alpha(G_t - \boldsymbol{x}(s_t)^T \boldsymbol{w})\boldsymbol{x}(s_t)$

$s_1 \quad [2\ 0\ 0\ 0\ 0\ 0\ 0\ 1] \cdot [1 \ldots 1]$

$X(s_1)^T w = 3$

$\Delta w = \alpha(0-3)[2\ 0\ 0\ 0\ 0\ 0\ 0\ 1] = -3\alpha X(s_1)$

Table of Contents

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'):

current estimate

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

- Target is $r + \gamma V^\pi(s')$
- In value function approximation, target is $r + \gamma \hat{V}^\pi(s'; \mathbf{w})$
- 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)

oracle: $\alpha (V^\pi - \hat{V}(s,w)) \nabla_w \hat{V}^\pi(s,w)$

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

- Note: we treat $\hat{V}^\pi(s'; \boldsymbol{w})$ in target as a **scalar** (it is a function of $\boldsymbol{w}$ but weight update ignores that)

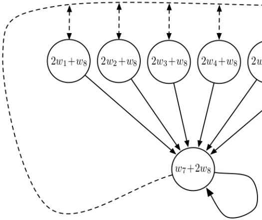
TD(0) Linear Value Function Approximation for Policy Evaluation

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

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

  $k = k + 1$
end loop

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



- $\mathbf{x}(s_1) = [2\ 0\ 0\ 0\ 0\ 0\ 0\ 1]$ $\mathbf{x}(s_2) = [0\ 2\ 0\ 0\ 0\ 0\ 0\ 1]$ $\ldots \mathbf{x}(s_6) = [0\ 0\ 0\ 0\ 0\ 2\ 0\ 1]$ $\mathbf{x}(s_7) = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 2]$ $r(s) = 0\ \forall s$ 2 actions a_1 solid line, a_2 dotted
- Small prob s_7 goes to terminal state s_T
- Consider tuple $(s_1, a_1, 0, s_7)$.
- Let w_0 =[1 1 1 1 1 1 1 1]. TD update: $\Delta \mathbf{w} = \alpha(r + \gamma \mathbf{x}(s')^T \mathbf{w} - \mathbf{x}(s)^T \mathbf{w})\mathbf{x}(s)$

$\gamma = 1$

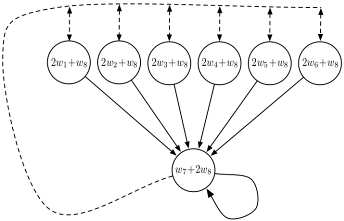
$X(s_1) w = 3$ $\quad 0 + \gamma X(s_7) w$

$\Delta w = \alpha(3\gamma - 3) X(s_1)$ TD 3

$\alpha(0 - 3) X(s_1)$ MC

[1] Figure from Sutton and Barto 2018

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



- $\boldsymbol{x}(s_1) = [2\ 0\ 0\ 0\ 0\ 0\ 0\ 1]$ $\boldsymbol{x}(s_2) = [0\ 2\ 0\ 0\ 0\ 0\ 0\ 1]$ $\ldots \boldsymbol{x}(s_6) = [0\ 0\ 0\ 0\ 0\ 2\ 0\ 1]$ $\boldsymbol{x}(s_7)= [0\ 0\ 0\ 0\ 0\ 0\ 1\ 2]$ $r(s) = 0\ \forall s$ 2 actions a_1 solid line, a_2 dotted
- Small prob s_7 goes to terminal state s_T
- Consider tuple $(s_1, a_1, 0, s_7)$.
- Let $w_0 =[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$. TD update: $\Delta \boldsymbol{w} = \alpha(r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w} - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}(s)$
- TD target is $r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w}$. $r = 0$ $\boldsymbol{x}(s')^T \boldsymbol{w} = 3$.
- $\boldsymbol{x}(s)^T \boldsymbol{w} = 3$
- $\Delta \boldsymbol{w} = \alpha(3\gamma - 3)\boldsymbol{x}(s_1)$

[1] Figure from Sutton and Barto 2018

Table of Contents

Control using Value Function Approximation

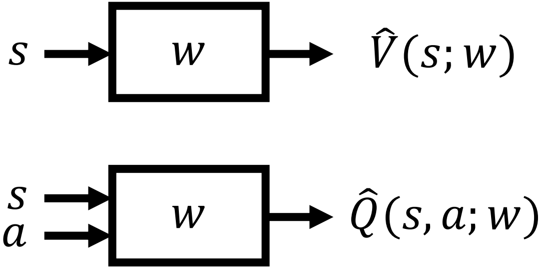
- Use value function approximation to represent state-action values $\hat{Q}^{\pi}(s, a; \boldsymbol{w}) \approx Q^{\pi}$
- Interleave
  - Approximate policy evaluation using value function approximation Q
  - Perform ϵ-greedy policy improvement
- Can be unstable. Generally involves intersection of the following:
  - Function approximation
  - Bootstrapping
  - **Off-policy learning**

} deadly triad

Control with VFA

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

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



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

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

- Stochastic gradient descent (SGD) samples the gradient

Check Your Understanding L5N2: 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 match the right weight update equations for the different methods: SARSA control update, Q-learning control update and MC control update.

$$
\begin{aligned}
\Delta \boldsymbol{w} &= \alpha(r + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})(1) \\
\Delta \boldsymbol{w} &= \alpha(G_t + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})(2) \\
\Delta \boldsymbol{w} &= \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})(3) \\
\Delta \boldsymbol{w} &= \alpha(G_t - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})(4) \\
\Delta \boldsymbol{w} &= \alpha(r + \gamma \max_{s'} \hat{Q}(s', a; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})(5)
\end{aligned}
$$

Check Your Understanding L5N2: 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
  $\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$
- (3) is the Q-learning control update
  $\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})(3)$
- (4) is the MC control update
  $\Delta \boldsymbol{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$

Linear State Action Value Function Approximation with an Oracle

- Use features to represent both the state and action

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

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

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

- Stochastic gradient descent update:

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

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

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

x(s, a) ← for linear

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

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

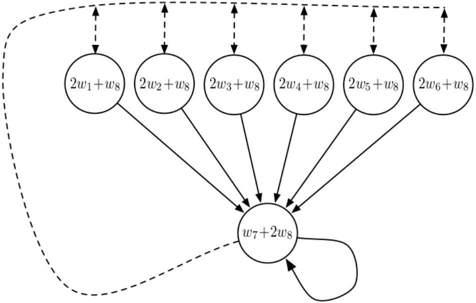
x(s, a)

- 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

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

x(s, a)

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

Geoff Gordon ~1995
averagers

Check Your Knowledge

- In TD learning with linear VFA (select all):
  1. $\mathbf{w} = \mathbf{w} + \alpha(r(s_t) + \gamma \mathbf{x}(s_{t+1})^T \mathbf{w} - \mathbf{x}(s_t)^T \mathbf{w})\mathbf{x}(s_t)$
  2. $V(s) = \mathbf{w}(s)\mathbf{x}(s)$
  3. Not sure

Check 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. Not sure

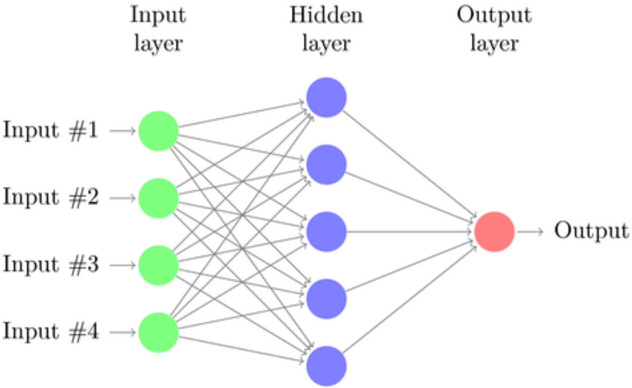
Answer: 1 is true. Convergence is not guaranteed to the best, the resulting one may still be worse than the best MSE solution by a factor of $\frac{1}{1-\gamma}$. It is also important to know that this is with respect to the stationary distirbution $d(s)$. Also note the weights do not depend on the state.

Table of Contents

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 [3]



[3] Figure by Kjell Magne Fauske

The Benefit of Deep Neural Network Approximators

- 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

Table of Contents

Deep Reinforcement Learning

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

Model-Free Control with General Function Approximators

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- Similar to linear value function approximation, but gradient with respect to complex function
- Monte Carlo: use return G_t as target

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

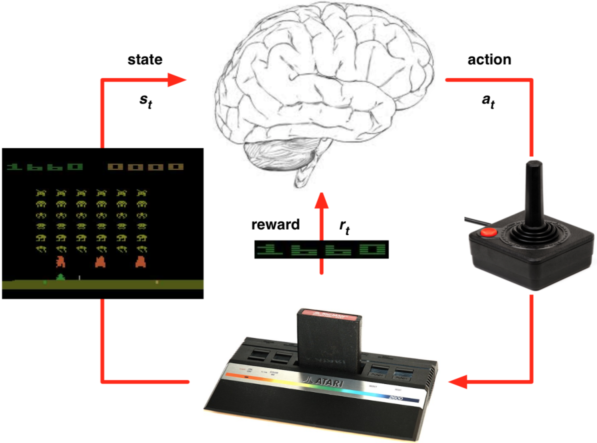
- SARSA: use a TD target $r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \boldsymbol{w})$, with current function approximation value

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

- For Q-learning

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

Using these ideas to do Deep RL in Atari



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
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow$ 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
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow$ 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

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

- **Uses target as a scalar, but function 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

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

$\approx Q^*(s,a)$

DQN Pseudocode

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

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
- 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
- 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

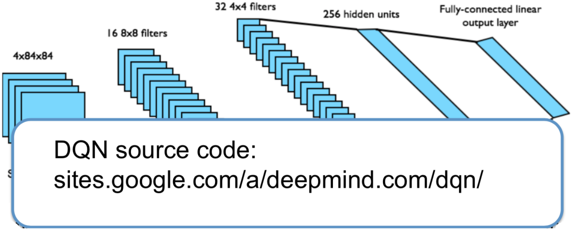
Answer: It doubles the memory requirements.

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

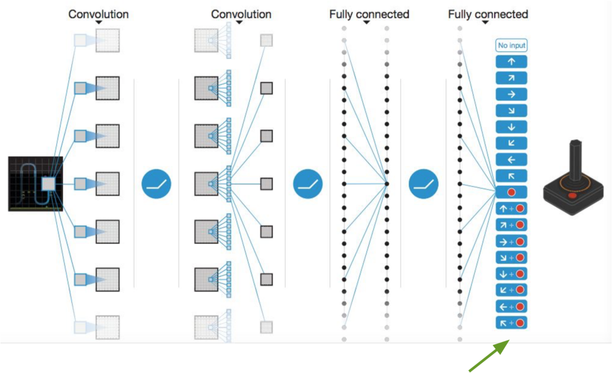
DQNs in Atari

- 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

DQN



1 network, outputs Q value for each action

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

DQN Results in Atari

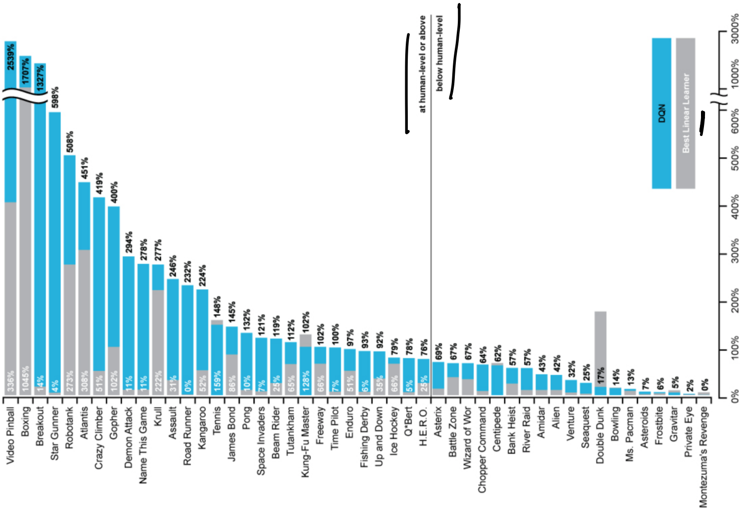


Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

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)

What You Should Understand

- Be able to implement TD(0) and MC on policy evaluation with linear value function approximation
- 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
- Be able to implement DQN and know some of the key features that were critical (experience replay, fixed targets)

Class Structure

- Last time and start of this time: Model-free reinforcement learning with function approximation
- Next time: Deep RL continued

Batch Monte Carlo Value Function Approximation

- May have a set of episodes from a policy π
- Can analytically solve for the best linear approximation that minimizes mean squared error on this data set
- Let $G(s_i)$ be an unbiased sample of the true expected return $V^\pi(s_i)$

$$\arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} (G(s_i) - \boldsymbol{x}(s_i)^T \boldsymbol{w})^2$$

- Take the derivative and set to 0

$$\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{G}$$

- where $\boldsymbol{G}$ is a vector of all N returns, and X is a matrix of the features of each of the N states $\boldsymbol{x}(s_i)$
- Note: not making any Markov assumptions

For next class

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:

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

Convergence Guarantees for Linear Value Function Approximation for Policy Evaluation

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

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

- 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 min mean squared error possible given distribution d:

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

Check Your Understanding L5N1: Poll

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

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

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

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

Check Your Understanding L5N1 : Poll

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

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

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

MSVE = 0 for TD

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

Convergence of Control Methods with VFA

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