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

- A TD error $> 0$ can occur even if the current $V(s)$ is correct $\forall s$: [select all]
A note on Monte Carlo vs TD estimates

- MC VFA
- Temporal Difference (TD(0)) Learning with Value Function Approximation
- Deep Q Learning
Policy evaluation: $\hat{V}^\pi \leftarrow (1 - \alpha)\hat{V}^\pi + \alpha V_{\text{target}}$

MC: $V_{\text{target}}(s_t) = G_t$ (sum of discounted returns until the episode terminates)
  - Target is unbiased estimate of $V^\pi$
  - Target can be high variance

TD(0): $V_{\text{target}}(s_t) = r_t + \gamma \hat{V}(s')$
  - Target is a biased estimate of $V^\pi$
  - Target is lower variance

Which one should we use? Is there other alternatives?
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 \( V^{\pi} \)
  - Target can be high variance
- **TD(0):** \( V_{target}(s_t) = r_t + \gamma \hat{V}(s') \)
  - Target is a biased estimate of \( V^{\pi} \)
  - Target is lower variance
- **Best of both worlds?**
- **n-step TD:** \( V_{target}(s_t) = r_t + \gamma r_{t+1} + \gamma r_{t+2} + \ldots \gamma^n \hat{V}(s_{t+n}) \)
Performance of n-step TD methods as a function of $\alpha$

Average RMS error over 19 states and first 10 episodes

---

1 Figure 7.2 from Sutton and Barto 2018

19 state random walk task.
2 Value Function Approximation

- MC VFA
- Temporal Difference (TD(0)) Learning with Value Function Approximation
- Deep Q Learning
Feature Vectors

- Use a feature vector to represent a state $s$

$$\mathbf{x}(s) = \begin{pmatrix} x_1(s) \\ x_2(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$
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
  \[
  \hat{V}(s; \mathbf{w}) = \sum_{j=1}^{n} x_j(s)w_j = \mathbf{x}(s)^T \mathbf{w}
  \]

- Objective function is
  \[
  J(\mathbf{w}) = \mathbb{E}_\pi[(V^\pi(s) - \hat{V}(s; \mathbf{w}))^2]
  \]

- Recall weight update is
  \[
  \Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})
  \]

- Update is:
  \[
  \Delta \mathbf{w} = -\frac{1}{2}\alpha (V^\pi(s) - \mathbf{x}(s)^T \mathbf{w}) \mathbf{x}
  \]

- Update = step-size × prediction error × feature value
Value Function Approximation

- MC VFA
- Temporal Difference (TD(0)) Learning with Value Function Approximation
- Deep Q Learning
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

$$\Delta w = \alpha (G_t - \hat{V}(s_t; w)) \nabla_w \hat{V}(s_t; w)$$

$$= \alpha (G_t - \hat{V}(s_t; w)) x(s_t)$$

$$= \alpha (G_t - x(s_t)^T w) x(s_t)$$

- 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 $\pi$

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: $\Delta w = \alpha (G_t - x(s_t)^T w)x(s_t)$

8: end if

9: end for

10: $k = k + 1$

11: end loop
Baird (1995)-Like Example with MC Policy Evaluation

\[ x(s_1) = [2 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1] \ x(s_2) = [0 \ 2 \ 0 \ 0 \ 0 \ 0 \ 1] \ \ldots \ x(s_6) = [0 \ 0 \ 0 \ 0 \ 2 \ 0 \ 1] \]
\[ x(s_7) = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 2] \quad r(s) = 0 \ \forall s \quad 2 \text{ actions } a_1 \text{ solid line, } a_2 \text{ dotted} \]

- Small prob \( s_7 \) goes to terminal state \( s_T \)
- Consider trajectory \((s_1, a_1, 0, s_7, a_1, 0, s_7, a_1, 0, s_T)\). \( G(s_1) = 0 \)
- Let \( w_0 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \). MC update: \( \Delta w = \alpha(G_t - x(s_t)^T w)x(s_t) \)
2 Value Function Approximation

- MC VFA
- Temporal Difference (TD(0)) Learning with Value Function Approximation
- Deep Q Learning
Temporal Difference (TD(0)) Learning with Value Function Approximation

- Uses bootstrapping and sampling to approximate true $V^\pi$
- 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 $r + \gamma V^\pi(s')$
- In value function approximation, target is $r + \gamma \hat{V}^\pi(s'; w)$
- 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 \( r + \gamma \hat{V}^{\pi}(s'; w) \), 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: 
  \[ \langle s_1, r_1 + \gamma \hat{V}^{\pi}(s_2; w) \rangle, \langle s_2, r_2 + \gamma \hat{V}(s_3; w) \rangle, \ldots \]
- Find weights to minimize mean squared error

\[
J(w) = \mathbb{E}_\pi[(r_j + \gamma \hat{V}^{\pi}(s_{j+1}, w) - \hat{V}(s_j; w))^2]
\]
In value function approximation, target is \( r + \gamma \hat{V}^\pi(s'; w) \), a biased and approximated estimate of the true value \( V^\pi(s) \).

Supervised learning on a different set of data pairs:
\[ \langle s_1, r_1 + \gamma \hat{V}^\pi(s_2; w) \rangle, \langle s_2, r_2 + \gamma \hat{V}(s_3; w) \rangle, \ldots \]

In linear TD(0)
\[
\Delta w = \alpha (r + \gamma \hat{V}^\pi(s'; w) - \hat{V}^\pi(s; w)) \nabla_w \hat{V}^\pi(s; w) \\
= \alpha (r + \gamma \hat{V}^\pi(s'; w) - \hat{V}^\pi(s; w)) x(s) \\
= \alpha (r + \gamma x(s')^T w - x(s)^T w) x(s)
\]

Note: we treat \( \hat{V}^\pi(s'; w) \) in target as a scalar (it is a function of \( w \) but weight update ignores that)
TD(0) Linear Value Function Approximation for Policy Evaluation

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

\[
w = w + \alpha \left( \sum \left( r + \gamma x(s')^T w - x(s)^T w \right) x(s) \right)
\]
5: \( k = k + 1 \)
6: end loop
Baird Example with TD(0) On Policy Evaluation

$x(s_1) = [2 0 0 0 0 0 1] \ x(s_2) = [0 2 0 0 0 0 1] \ldots \ x(s_6) = [0 0 0 0 2 0 1] \ x(s_7) = [0 0 0 0 0 1 2] \ r(s) = 0 \ \forall s \ \ \ \ \text{2 actions } a_1 \text{ solid line, } a_2 \text{ dotted}$

Small prob $s_7$ goes to terminal state $s_T$

Consider tuple $(s_1, a_1, 0, s_7)$.

Let $w_0 = [1 1 1 1 1 1 1]$. TD update: $\Delta w = \alpha(r + \gamma x(s')^T w - x(s)^T w)x(s)$

---

$^1$Figure from Sutton and Barto 2018
x(s_1) = [2 0 0 0 0 0 1] x(s_2) = [0 2 0 0 0 0 1] ... x(s_6) = [0 0 0 0 0 2 0 1] 
\[ x(s_7) = [0 0 0 0 0 0 1 2] \quad r(s) = 0 \quad \forall s \quad 2 \text{ actions } a_1 \text{ solid line, } a_2 \text{ dotted} \]

- Small prob \( s_7 \) goes to terminal state \( s_T \)
- Consider tuple \((s_1, a_1, 0, s_7)\).
- Let \( w_0 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \). TD update: \( \Delta w = \alpha (r + \gamma x(s')^T w - x(s)^T w) x(s) \)
Table of Contents

- MC VFA
- Temporal Difference (TD(0)) Learning with Value Function Approximation

3 Control using Value Function Approximation
  - Deep Q Learning
Control using Value Function Approximation

- Use value function approximation to represent state-action values
  \[ \hat{Q}^\pi(s, a; w) \approx Q^\pi \]

- Interleave
  - Approximate policy evaluation using value function approximation
  - Perform \(\epsilon\)-greedy policy improvement

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

- Represent state-action value function by Q-network with weights $\mathbf{w}$
  $$\hat{Q}(s, a; \mathbf{w}) \approx Q(s, a)$$
Action-Value Function Approximation with an Oracle

- \( \hat{Q}^\pi(s, a; w) \approx Q^\pi \)

- Minimize the mean-squared error between the true action-value function \( Q^\pi(s, a) \) and the approximate action-value function:

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

- Use stochastic gradient descent to find a local minimum

\[
\Delta(w) = \alpha \nabla_w J(w) = \alpha \mathbb{E} \left[(Q^\pi(s, a) - \hat{Q}^\pi(s, a; w)) \nabla_w \hat{Q}^\pi(s, a; w) \right]
\]

- 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.

\[ \Delta w = \alpha (r + \gamma \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}(s, a; w) \] (1)

\[ \Delta w = \alpha (G_t + \gamma \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}(s, a; w) \] (2)

\[ \Delta w = \alpha (r + \gamma \max_{a'} \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}(s, a; w) \] (3)

\[ \Delta w = \alpha (G_t - \hat{Q}(s_t, a_t; w)) \nabla_w \hat{Q}(s_t, a_t; w) \] (4)

\[ \Delta w = \alpha (r + \gamma \max_{s'} \hat{Q}(s', a; w) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}(s, a; w) \] (5)
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.
Linear State Action Value Function Approximation with an Oracle

- Use features to represent both the state and action

\[ x(s, a) = \begin{pmatrix} x_1(s, a) \\ x_2(s, a) \\ \vdots \\ x_n(s, a) \end{pmatrix} \]

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

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

- Stochastic gradient descent update:

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

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

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

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

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

  $$\Delta w = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}(s, a; w)$$
Challenges of Off Policy Control: Baird Example

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

\[ \pi(\text{solid}|\cdot) = 1 \]
\[ \mu(\text{dashed}|\cdot) = \frac{6}{7} \]
\[ \mu(\text{solid}|\cdot) = \frac{1}{7} \]
\[ \gamma = 0.99 \]
In TD learning with linear VFA (select all):

1. \[ w = w + \alpha (r(s_t) + \gamma x(s_{t+1})^T w - x(s_t)^T w)x(s_t) \]
2. \[ V(s) = w(s)x(s) \]
3. Not sure
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. \( V(s) = \mathbf{w}(s)^T \mathbf{x}(s) \)
3. Not sure
Table of Contents

- MC VFA
- Temporal Difference (TD(0)) Learning with Value Function Approximation

Deep learning for Value Function Approximation
- Deep Q Learning
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.
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

- MC VFA
- Temporal Difference (TD(0)) Learning with Value Function Approximation

4 Deep learning for Value Function Approximation
  - Deep Q Learning
Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value, Q function
  - Policy
  - Model

- Optimize loss function by stochastic gradient descent (SGD)
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

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

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

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

For Q-learning

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

$$
\begin{align*}
s_1, a_1, r_2, s_2 \\
\vdots \\
\vdots \\
s_t, a_t, r_{t+1}, s_{t+1}
\end{align*}
\rightarrow s, a, r, s'
$$

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

$$
\Delta w = \alpha (r + \gamma \max_{a'} \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}(s, a; w)
$$
DQNs: Experience Replay

- To help remove correlations, store dataset $D$ from prior experience
  
  $D = \{s_1, a_1, r_2, s_2, s_2, a_2, r_3, s_3, \ldots, s_t, a_t, r_{t+1}, s_{t+1}\}$

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

- Uses target as a scalar, but function weights will get updated on the next round, changing the target value
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 $\mathbf{w}^-$ be the set of weights used in the target, and $\mathbf{w}$ 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 $s$: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}^-)$
- Use stochastic gradient descent to update the network weights:

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

1: Input $C$, $\alpha$, $D = \{\}$, Initialize $\mathbf{w}$, $\mathbf{w}^- = \mathbf{w}$, $t = 0$
2: Get initial state $s_0$
3: loop
4: Sample action $a_t$ given $\epsilon$-greedy policy for current $\hat{Q}(s_t, a; \mathbf{w})$
5: Observe reward $r_t$ and next state $s_{t+1}$
6: Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $D$
7: Sample random minibatch of tuples $(s_i, a_i, r_i, s_{i+1})$ from $D$
8: for $j$ in minibatch do
9: if episode terminated at step $i + 1$ then
10: $y_i = r_i$
11: else
12: $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \mathbf{w}^-)$
13: end if
14: Do gradient descent step on $(y_i - \hat{Q}(s_i, a_i; \mathbf{w}))^2$ for parameters $\mathbf{w}$: $\Delta \mathbf{w} = \alpha (y_i - \hat{Q}(s_i, a_i; \mathbf{w})) \nabla_{\mathbf{w}} \hat{Q}(s_i, a_i; \mathbf{w})$
15: end for
16: $t = t + 1$
17: if mod$(t, C) == 0$ then
18: $\mathbf{w}^- \leftarrow \mathbf{w}$
19: end if
20: 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: 
\[ r + \gamma \max_{a'} \hat{Q}(s', a'; w^-) \]

- 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
In DQN we compute the target value for the sampled \((s, a, r, s')\) using a separate set of target weights: 

\[ r + \gamma \max_{a'} \hat{Q}(s', a'; w^-) \]

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
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 \(D\)
- Sample random mini-batch of transitions \((s, a, r, s')\) from \(D\)
- Compute Q-learning targets w.r.t. old, fixed parameters \(w^-\)
- 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 source code: sites.google.com/a/deepmind.com/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?

<table>
<thead>
<tr>
<th>Game</th>
<th>Linear</th>
<th>Deep Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Enduro</td>
<td>62</td>
<td>29</td>
</tr>
<tr>
<td>River Raid</td>
<td>2345</td>
<td>1453</td>
</tr>
<tr>
<td>Seaquest</td>
<td>656</td>
<td>275</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>301</td>
<td>302</td>
</tr>
</tbody>
</table>

Note: just using a deep NN actually hurt performance sometimes!
Which Aspects of DQN were Important for Success?

<table>
<thead>
<tr>
<th>Game</th>
<th>Linear</th>
<th>Deep Network</th>
<th>DQN w/ fixed Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>3</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Enduro</td>
<td>62</td>
<td>29</td>
<td>141</td>
</tr>
<tr>
<td>River Raid</td>
<td>2345</td>
<td>1453</td>
<td>2868</td>
</tr>
<tr>
<td>Seaquest</td>
<td>656</td>
<td>275</td>
<td>1003</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>301</td>
<td>302</td>
<td>373</td>
</tr>
</tbody>
</table>
## Which Aspects of DQN were Important for Success?

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

- **Replay is ****hugely** important
- Why? Beyond helping with correlation between samples, what does replaying do?
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)
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)
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 $\pi$
- 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 $V^\pi(s_i)$

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

- Take the derivative and set to 0

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

- where $G$ is a vector of all $N$ returns, and $X$ is a matrix of the features of each of the $N$ states $x(s_i)$
- Note: not making any Markov assumptions
For next class
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 $\pi$
- $\sum_s d(s) = 1$
- $d(s)$ satisfies the following balance equation:

$$d(s') = \sum_s \sum_a \pi(a|s)p(s'|s, a)d(s)$$
Define the mean squared error of a linear value function approximation for a particular policy $\pi$ relative to the true value given the distribution $d$ as

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

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

TD(0) policy evaluation with VFA converges to weights $w_{TD}$ which is within a constant factor of the min mean squared error possible given distribution $d'$:

$$MSVE_d(w_{TD}) \leq \frac{1}{1 - \gamma} \min_w \sum_{s \in S} d(s) (V^\pi(s) - \hat{V}^\pi(s; w))^2$$
TD(0) policy evaluation with VFA converges to weights $w_{TD}$ which is within a constant factor of the min mean squared error possible for distribution $d$:

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

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


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

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

- If the VFA is a tabular representation (one feature for each state), what is the $MSVE_d$ for TD?
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.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Tabular</th>
<th>Linear VFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monte-Carlo Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sarsa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q-learning</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>