Value Function Approximation
**Short Comings of the methods so far**

**So far:** All methods work on a discrete state space $S$.

⇒ A policy $\pi$ is a table of the form $\{(s_1, a_1), \ldots, (s_{|S|}, a_{|S|})\}$

⇒ If we encounter a new state, we do not know what to do.

⇒ No matter how similar two states are, we learn $Q(s,a)$ independently.

⇒ If $|S|$ is very large:
  - We need a lot of memory to store the policy.
  - We need enormous amounts of samples to estimate $Q(s,a)$ for all state-action pairs.

⇒ Previous models for MDPs and Reinforcement learning become infeasible.
Some examples

- Number of states for some problems
  - Backgammon: $10^{20}$
  - Computer Go: $10^{170}$
  - Flying an RC Helicopter: continuous state space
Working with continuous State Spaces

**Idea:** What if we do not distinguish states but state descriptions, e.g., feature vectors?

- Depending on the feature space we can describe an infinite set of states. But some states might have the same description.
  
  \[ \Rightarrow \text{c.f. we often work on observations not states anyway} \]

- A policy can be described as a function \( f \) of the state space
  
  \[ f(x,a) = Q(s,a) \text{ or } f(x) = a \]
  
  \[ \Rightarrow \text{Mathematical functions are much more space efficient than tables} \]

- State descriptions can be related to each other \( \Rightarrow \) if we do not have encountered a particular state description so far, we can derive a proper action from similar situations. (generalization)

**Generally:** Working on state descriptions allows for flexible agents being able to cope with unknown situations.
Overview on continuous State Spaces

• Value function approximation (this lecture)
  – Learn a function $f$ to predict $U(x_s)$ or $Q(x_s, a)$
    (generally $f$ is a regression function of some kind)

• Policy gradient methods: (next lecture)
  – Directly learn a function $f(x_s)$ predicting the best action $a$ for $x_s$

• Actor Critic methods: (next lecture)
  – combine policy functions and value function approximation
**Value function approximation**

**Given**: A mapping \( x(s) \) describing \( s \) in \( IR^d \).

**Idea**: Learn a function that either describes the utility \( U(S) \) or the state-value function \( Q(S, A) \).

Options to learn the \( f(s, \theta) \approx U(S) \) or \( f(s, a, \theta) \approx Q(S, A) \):

- **Approximate \( U(S) \)**

  \[
  x(s) \xrightarrow{\theta} f(s, \theta)
  \]

- **Approximate \( Q(S, A) \)**

  \[
  x(s) \xrightarrow{a \quad \theta} f(s, a, \theta)
  \]

  or

  \[
  x(s) \xrightarrow{\theta} f(s, a_1, \theta) \quad : \quad f(s, a_i, \theta)
  \]
Value Function Approximation and Partial Observability

A side-effect of using value function approximation is that we can work on a factor space representing the exact state S or just an observation O.

- factor spaces: often the state can be coded as a set of (independent) parameters:
  Example: position of the agent + state variables of the environment, Stockmarket: recent course development for all traded stocks, ...
- Observation spaces: a set of parameters giving us hints about the state.
  Examples: video buffer of a camera, sensor data, player view in a video game,

⇒ Since \( f(x(s),a, \theta) \) is an approximation function works for both settings (\( f(x(s),a, \theta) \) can learn to consider belief states)

⇒ **Caution:** Make sure that \( x(s) \) is Markov !!!
Mean Squared Value Error

Regardless of how we built our approximation function $f(S, \theta)$, we need a measure for the quality of an approximation:

$$
\overline{VE}^\pi(\theta) = \mathbb{E}_{\pi, S \sim \mu} \left[ (U_\pi(S) - f(S, \theta))^2 \right]
$$

$$
= \sum_{s \in S} \mu(S) (U_\pi(S) - f(S, \theta))^2
$$

where $\mu$ is the importance distribution over the state descriptions with $\sum_{s \in S} \mu(S)$.
For example, we can take $\mu(S)$ as the likelihood of being in state $s$ when following $\pi$. 

Common types of function approximators

- Generally any regression/prediction function can be used (usually we will require a continuous return to model the Utility)
- Common methods:
  - Linear predictors
  - Neural networks
  - Decision trees
  - Regression with Fourier/Wavelet bases
  - ..
- However: Reinforcement learning is tricky because:
  - experience is non-stationary (e.g. the label Q(S,A) might change when using TD learning)
  - experience is usually non-iid the observation from a single episode is usually highly correlated
**Value Function Approx. with SGD**

**Goal:** Given policy $\pi$ and $U_\pi(S)$ find $\theta$ minimizing a loss function $L^\pi(\theta)$.

Note: We won’t have $U_\pi(S)$ but only $R(S)$ later on.

For example, consider $L_{X,Y}(\theta)$ is mean square loss:

$$L^\pi(\theta) = \mathbb{E}_\pi [(U_\pi(S) - f(S, \theta))^2]$$

Computing the gradient we get

$$\Delta \theta = -\frac{1}{2} \alpha \nabla_\theta L^\pi(\theta) = \alpha \mathbb{E}_\pi [(U_\pi(S) - f(S, \theta)) \nabla_\theta L(S, \theta)]$$

- With SGD we sample the gradient:

  $$\Delta \theta = \alpha (U_\pi(S) - f(S, \theta)) \nabla_\theta L(S, \theta)$$

- the expected update is equal to the full gradient update
Linear Prediction Functions

A simple function approximation might be linear.

• Linear Functions over $x(S) \in \mathbb{R}^d$ where $\theta$ is a weight vector $w$:
  
  $$f(x(S), W) = x(S)^T W = \sum_{j=1}^{n} x(S)_j w_j$$

• Loss function:
  
  $$L(W) = E[(U(s) - x(s)^T W)^2]$$

• Stochastic Gradient Descent on $L(w)$:
  
  $$\nabla W f(x(s), W) = x(s)$$
  
  $$- \frac{1}{2} \nabla L(\theta) = (U(s) - f(x(s), \theta)) x(s)$$
  
  $$\Delta \theta = \alpha (U(s) - f(x(s), \theta)) x(s)$$

  update = step size $\times$ prediction error $\times$ feature vector
Table Lookup Features

• Table lookups can be considered as a special case of linear value function approximation

• Use a lookup table of the of the following form:

\[ x^{table}(S) = \begin{pmatrix} 1(S = s_1) \\ \vdots \\ 1(S = s_n) \end{pmatrix} \]

• Parameter vector \( w \) gives us the value of each state:

\[ f(x(S), w) = \begin{pmatrix} 1(S = s_1) \\ \vdots \\ 1(S = s_n) \end{pmatrix}^T \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} \]
Incremental Prediction algorithms

- In practice we do not have the utility \( U_\pi(S) \) but only \( R(S) \)
  \( \Rightarrow \) We have to employ a target for \( U_\pi(S) \) as in the last lecture

Prediction based on value function approximation:
- For MC, the target is the complete return \( G_t \)
  \[
  \Delta \theta = \alpha (G_t - f(x(S_t), \theta)) \nabla_\theta f(x(S_t), \theta)
  \]
- For TD, the target is the TD target \( R_{t+1} + \gamma f(x(S_{t+1}), \theta) \)
  \[
  \Delta \theta = \alpha (R_{t+1} + \gamma f(x(S_{t+1}), \theta) - f(x(S_t), \theta)) \nabla_\theta f(x(S_t), \theta)
  \]
- For TD(\( \lambda \)), the target is the \( \lambda \)-return \( G_t^\lambda \)
  \[
  \Delta \theta = \alpha (G_t^\lambda - f(x(S_t), \theta)) \nabla_\theta f(x(S_t), \theta)
  \]

**Caution:** For TD and TD(\( \lambda \)) the target depends on \( \theta \)
\( \Rightarrow \) TD and TD(\( \lambda \)) are semi-gradient methods because the gradient is only computed w.r.t. \( f(x(S_t), \theta) \), but the for the target functions.
MC with value function approximation

• Return $G_t$ is an unbiased, noisy sample of true value $U(S)$
• Applying supervised learning to known experience is viable:
  $(x(S_1), G_1), (x(S_2), G_2), ... (x(S_T), G_T)$

• For example, linear Monte-Carlo policy evaluation:
  \[
  \Delta \theta = \alpha (G_t - f(x(S_t), \theta)) \nabla_{\theta} f(x(S_t), \theta) \\
  = \alpha (G_t - f(x(S_t), \theta)) \cdot x(S_t)
  \]

• Monte-Carlo evaluation converges to a local optimum

• Even when using non-linear value function approximation
TD with value function approximation

- The TD-target is a biased sample sample of true value $U(S)$
- Applying supervised learning is still possible but training data looks like:
  \[
  (x(S_1), R_1 + \gamma f(x(S_2), \theta)), (x(S_1), R_2 + \gamma f(x(S_3), \theta)), \ldots (x(S_{T-1}), R_T)
  \]
- For example, linear TD(0) policy evaluation:
  \[
  \Delta \theta = \alpha (R_t + \gamma f(x(S_{t+1}), \theta) - f(x(S_t), \theta)) \nabla_{\theta} f(x(S_t), \theta)
  = \alpha (R_t + \gamma f(x(S_{t+1}), \theta) - f(x(S_t), \theta)) \cdot x(S_t)
  = \alpha \delta \cdot x(S_t)
  \]
- Linear TD(0) converges (close) to global optimum
• The $\lambda$-return is also a biased sample sample of true value $U(S)$
• Applying supervised learning is to training data of the form:
  $(x(S_1), G_1^\lambda), (x(S_2), G_2^\lambda), \ldots, (x(S_{T-1}), G_{T-1}^\lambda)$

• Forward view of linear TD($\lambda$):
  \[
  \Delta \theta = \alpha \left( G_1^\lambda - f(x(S_t), \theta) \right) \nabla \theta f(x(S_t), \theta)
  = \alpha \left( G_1^\lambda - f(x(S_t), \theta) \right) \cdot x(S_t)
  \]

• Forward view of linear TD($\lambda$):
  \[
  \delta_t = R_{t+1} + \gamma f(x(S_{t+1}), \theta) - f(x(S_t), \theta)
  E_t = \gamma \lambda E_{t-1} + x(S_t)
  = \alpha \delta \cdot E_t
  \]
## Convergence of Prediction Methods

<table>
<thead>
<tr>
<th>On/Off policy</th>
<th>Algorithm</th>
<th>Table Lookup</th>
<th>Linear</th>
<th>Non-Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Policy</td>
<td>MC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>On-Policy</td>
<td>TD</td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>On-Policy</td>
<td>TD((\lambda))</td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Off-Policy</td>
<td>MC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Off-Policy</td>
<td>TS</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>Off-Policy</td>
<td>TD((\lambda))</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
</tr>
</tbody>
</table>
Control and Value Function Approximation

- To apply policy iteration, we again have to switch to state-value functions $Q(S,A)$

- Basic idea for on-policy learning:
  - approximate $q_\pi$ with a function $q(x(S),a,\theta)$:
    \[ \hat{q}(S,A,\theta) \approx q_\pi(S,A) \]
  - employ $\epsilon$-greedy policy improvement

Caution:
- It is not necessary to approximate $q_\pi(S,A)$ very accurately. Instead, we take a step into improving $\hat{q}(S,A,\theta)$ and then adjust the policy.
- Using function approximation is not guaranteed to converge against $q_\pi(S,A)$. Since $\hat{q}(S,A,\theta)$ is a regression function it is not guaranteed that the model can describe the real $q_\pi(S,A)$ for all $(S,A)$.  

Control and Value Function Approximation

To learn a reasonable close $\hat{q}(S, A, \theta)$, we can:

• Minimize the mean square error between the approximation $\hat{q}(S, A, \theta)$ and the true action value $q_\pi(S, A)$:

$$L(\theta) = \mathbb{E}_{\pi, S \sim \mu} \left[ (q_\pi(s, a) - \hat{q}(s, a, \theta))^2 \right]$$

• Optimization via SGD:

$$- \frac{1}{2} \nabla_\theta L(\theta) = (q_\pi(S, A) - \hat{q}(S, A, \theta)) \nabla_\theta \hat{q}(S, A, \theta)$$

$$\Delta \theta = \alpha (q_\pi(S, A) - \hat{q}(S, A, \theta)) \nabla_\theta \hat{q}(S, A, \theta)$$
Control with Linear Value Functions

- State-action are modelled as a feature vector:
  \[ x(S, A) = \begin{pmatrix} x_1(S, A) \\ \vdots \\ x_n(S, A) \end{pmatrix} \]

- Represent action-value function by linear combination of features
  \[ \hat{q}(S, A, w) = \sum_{j=1}^{n} x_j(S, A)w_j \]

- With the SGD update:
  \[
  -\frac{1}{2} \nabla_w L(w) = (q_\pi(S, A) - (x(S, A)^T w)) \nabla_w (x(S, A)^T w) \\
  = (q_\pi(S, A) - (x(S, A)^T w))w \\
  \Delta w = \alpha (q_\pi(S, A) - \hat{q}(S, A, w)) x(S, A)
  \]
similar to prediction but substitute \( q_\pi(S, A) \):

- For MC, the target is the complete return \( G_t \)
  \[
  \Delta \theta = \alpha (G_t - \hat{q}(S_t, A_t, \theta)) \nabla_{\theta} \hat{q}(S_t, A_t, \theta)
  \]

- For TD, the target is the TD target \( R_{t+1} + \gamma \hat{q}(S_t, A_t, \theta) \)
  \[
  \Delta \theta = \alpha (R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \theta) - \hat{q}(S_t, A_t, \theta)) \nabla_{\theta} \hat{q}(S_t, A_t, \theta)
  \]

- For TD(\( \lambda \)), the target is the action-value \( \lambda \)-return \( q_t^\lambda \):
  - Forward view
    \[
    \Delta \theta = \alpha \left( q_t^\lambda - \hat{q}(S_t, A_t, \theta) \right) \nabla_{\theta} \hat{q}(S_t, A_t, \theta)
    \]
  - Backward view:
    \[
    \delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \theta) - \hat{q}(S_t, A_t, \theta)
    \]
    \[
    E_t = \lambda \gamma E_{t-1} + \nabla_{\theta} \hat{q}(S_t, A_t, \theta)
    \]
    \[
    \Delta \theta = \alpha \delta_t E_t
    \]
Example: Mountain Car

Semi-gradient Sarsa method with tile-coding

Figure 10.2: Mountain Car learning curves for the semi-gradient Sarsa method with tile-coding function approximation and "-greedy action selection.
Control and Convergence

• Convergence to the minimal error between $Q(S,A,\theta)$ and $q_{\pi}(S,A)$ is problematic.
• Generally convergence is problematic if we employ:
  • Value Function Approximation
  • Bootstraping
  • Off-Policy Learning
    (Deadly Triad)

=> For these cases, updates might even increase the error.
Baird’s Counterexample

- episodic MDP with 7 states and 2 actions:
  - Dashed: go to any of the upper states with 1/6
  - Solid: go to lower state 100%
- reward is always 0
  ⇒ true value functions = 0
- $\gamma=0.999$
- behavioural policy $b$:
  - $b(\text{dashed} | \cdot) = 6/7$
  - $b(\text{solid} | \cdot) = 1/7$
- target policy:
  - $\pi(\text{solid} | \cdot) = 1.0$

feature vectors:
$x(1)=(2,0,0,0,0,0,0,1)^T$
$x(2)=(0,2,0,0,0,0,0,1)^T$
...
$x(7)=(0,0,0,0,0,0,1,2)^T$

Applying semi-gradient TD(0) makes the weights diverge into infinity but switching to on-policy makes the TD(0) converge.
Making gradient methods converge

- TD is not a full GD approach
- Idea: Compute the complete gradient over.
  - Straight-forward the error function is smoothed rather then optimized
  - Gradient TD follows the true gradient of projected Bellman error and therefore does not diverge

<table>
<thead>
<tr>
<th>On/Off policy</th>
<th>Algorithm</th>
<th>Table Lookup</th>
<th>Linear</th>
<th>Non-Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Policy</td>
<td>MC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>On-Policy</td>
<td>TD</td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>On-Policy</td>
<td>Gradient TD</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Off-Policy</td>
<td>MC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Off-Policy</td>
<td>TS</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>Off-Policy</td>
<td>Gradient TD</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
# Convergence of Control Methods

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Table Lookup</th>
<th>Linear</th>
<th>Non-Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC Control</td>
<td>✔</td>
<td>(✔)</td>
<td>✘</td>
</tr>
<tr>
<td>Sarsa</td>
<td>✔</td>
<td>(✔)</td>
<td>✘</td>
</tr>
<tr>
<td>Q-Learning</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>Gradient Q-Learning</td>
<td>✔erals</td>
<td>✔</td>
<td>✘</td>
</tr>
</tbody>
</table>

(✔) = jumps around the near optimal value function
Batch Methods

Utilization of experience is rather bad with GD methods.

⇒ Batch Reinforcement Learning: Find the best fitting value function for the given experience ("training data")

⇒ Using an example only once for making one step might be a waste of experience
Least Squares Prediction

- Given the value function approximation \( f(s, \theta) \approx U(S) \)
- The experience \( \mathcal{D} \) is given by a set of state-value pairs
  \[ \mathcal{D} = \{(s_1, U^\pi(s_1)), \ldots, (s_T, U^\pi(s_T))\} \]

We want to find the parameters \( \theta \) to provide the value function approximation \( f(s, \theta) \) with the best fit on \( \mathcal{D} \).

\[ \Rightarrow \text{The least squares method fits the } \theta \text{ to minimize the sum-squared error between } f(s, \theta) \text{ and } U(S). \]

\[ LS_D(\theta) = \sum_{t=1}^{T} (U^\pi(s_t) - f(s, \theta))^2 \]

\[ \approx \mathbb{E}_{\mathcal{D}}[(U^\pi(s_t) - f(s, \theta))^2] \]
SGD with Experience Replay

Given experience $\mathcal{D}$ is given by a set of state-value pairs

$\mathcal{D} = \{(s_1, U^\pi(s_1)), \ldots, (s_T, U^\pi(s_T))\}$

Repeat:

- Sample state-value pair from $\mathcal{D}$:
  $\langle(s, U^\pi(s))\rangle \sim \mathcal{D}$

- Apply SGD update on the parameters $\theta$:
  $\Delta \theta = \alpha (U^\pi(s) - f(s, \theta)) \nabla_\theta f(s, \theta)$

$\Rightarrow$ Converges to least squares solution

$\theta^\pi = \arg \max_\theta LS_\mathcal{D} (\theta)$
Experience Replay in Deep Q-Networks (DQN)

- DQN applies deep learning to off-policy, non-linear, TD-target reinforcement learning. (danger of instability)
- by using experience replay and fixed Q-targets can be trained in a stable way.
- Experience:
  - observed transitions \((s_t, a_t, r_{t+1}, s_{t+1})\) in replay memory \(\mathcal{D}\).
  - By sampling independently from \(\mathcal{D}\) episodes are decoupled

- Idea of fixed Q-targets:
  - Q-learning targets in the experience replay are all generated w.r.t. “old”, fixed parameters \(\theta^-\).
  - Thus, Q-targets are independent from \(\theta\) in \(f(s, \theta)\) which are updated
DQN Algorithm

Repeat:
• Take action \( a_t \) according to \( \varepsilon \)-greedy policy
• Store transition \( (s_t, a_t, r_{t+1}, s_{t+1}) \) in replay buffer \( \mathcal{D} \)
• Sample random mini-batch of transition \( (s, a, r, s') \) from \( \mathcal{D} \)
• Compute the Q-learning targets w.r.t. fixed \( \theta^- \)
• Optimize MSE between Q-network and Q-learning targets:

\[
\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[ \left( r + \gamma \max_{a'} f(s', a'; \theta^-) - f(s, a, \theta) \right)^2 \right]
\]

by SGD
Example: DQN on Atari games

https://gym.openai.com/envs/#atari
Idea: Use one network architecture to learn multiple computer games on the video buffer as input.

- End-to-end learning of $Q(s,a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames (a single frame is not Markov!!)
- Actions: 18 Joystick/button combination (9 directions + 2 button states)
- Reward change in score for the step (most Atari games had general scores constantly rewarding actions)
DQN Results in Atari
Advantages replay buffer and fixed targets

<table>
<thead>
<tr>
<th>Game</th>
<th>Replay Fixed-Q</th>
<th>Replay Q-learning</th>
<th>No replay Fixed-Q</th>
<th>No replay Q-learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>316.81</td>
<td>240.73</td>
<td>10.16</td>
<td>3.17</td>
</tr>
<tr>
<td>Enduro</td>
<td>1006.3</td>
<td>831.25</td>
<td>141.89</td>
<td>29.1</td>
</tr>
<tr>
<td>River Raid</td>
<td>7446.62</td>
<td>4102.81</td>
<td>2867.66</td>
<td>1453.02</td>
</tr>
<tr>
<td>Seaquest</td>
<td>2894.4</td>
<td>822.55</td>
<td>1003</td>
<td>275.81</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>1088.94</td>
<td>826.33</td>
<td>373.22</td>
<td>301.99</td>
</tr>
</tbody>
</table>

Lecture notes D. Silver: Introduction to Reinforcement Learning ([http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html](http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html))

Lecture 6: Function Approximation (Slide 41)
Linear Least Squares Prediction

- Experience replay finds the least-squares solutions by sampling and using SGD on mini-batches.
- If we use linear value function approximation, we can analytically solve for a solution minimizing $LS_D(\theta)$:
  \[
  LS_D(w) = \sum_{t=1}^{T} (U^\pi(s_t) - x_t^T w)^2
  \]
  \[
  \frac{\partial LS_D(w)}{\partial w} = 2 \sum_{t=1}^{T} (U^\pi(s_t) - x_t^T w)x_t
  \]
- For a quadratic loss the derivative is linear and to find a local minimum we have to compute its zero.

- Experience replay finds the least-squares solutions by sampling and using SGD on mini-batches.
- If we use linear value function approximation, we can analytically solve for a solution minimizing $LS_D(\theta)$:
  \[
  LS_D(w) = \sum_{t=1}^{T} (U^\pi(s_t) - x_t^T w)^2
  \]
  \[
  \frac{\partial LS_D(w)}{\partial w} = 2 \sum_{t=1}^{T} (U^\pi(s_t) - x_t^T w)x_t
  \]
- For a quadratic loss the derivative is linear and to find a local minimum we have to compute its zero.
Linear Least Squares Prediction

- At minimum $LS_D(w) = 0$, i.e., the expected update is zero as well.
  \[ E_D[\Delta w] = 0 \]
  \[
  \alpha \sum_{t=1}^{T} x(s_t)(U^\pi(s_t) - x(s_t)^T w) = 0
  \]
  \[
  \sum_{t=1}^{T} x(s_t) U^\pi(s_t) = \sum_{t=1}^{T} x(s_t)x(s_t)^T w
  \]
  \[
  w = \left( \sum_{t=1}^{T} x(s_t)x(s_t)^T \right)^{-1} \sum_{t=1}^{T} x(s_t) U^\pi(s_t)
  \]
- For N features, direct solution $O(N^3)$ (matrix inversion)
- Incremental solution time $O(N^2)$ using Sherman-Morrison
- Usability depends on the ratio between the number of features N and the number of samples T
Linear Least Squares Prediction Algorithms

- Again: in practice $U^{\pi}(s_t)$ is yet unknown
  ⇒ Use noisy or biased samples of $U^{\pi}(s_t)$

- LMSC - Least Squares Monte-Carlo
  
  $U^{\pi}(s_t) \approx G_t$

- LSTD – Least Squares Temporal Difference Learning
  
  $U^{\pi}(s_t) \approx R_{t+1} + \gamma f(s_t, w)$

- LSTD(\(\lambda\)) – Least Squares with TD(\(\lambda\))
  
  $U^{\pi}(s_t) \approx G^\lambda_t$

⇒ For each target we can solve directly for the fixed point of MC/TD/ TD(\(\lambda\)) because the targets are considered as fixed
Linear Least Squares Prediction Algorithms (2)

LMSC: \[ 0 = \alpha \sum_{t=1}^{T} (G_t - x(s_t)^T w) x(s_t) \]
\[ w = (\sum_{t=1}^{T} x(s_t) x(s_t)^T)^{-1} \sum_{t=1}^{T} x(s_t) G_t \]

LSTD: \[ 0 = \alpha \sum_{t=1}^{T} (R_{t+1} + \gamma x(s_{t+1})^T w - x(s_t)^T w) x(s_t) \]
\[ w = \left( \sum_{t=1}^{T} x(s_t)(x(s_t) - \gamma x(s_{t+1}))^T \right)^{-1} \sum_{t=1}^{T} x(s_t) R_{t+1} \]

LSTD(\(\lambda\)): \[ 0 = \sum_{t=1}^{T} \alpha \delta_t E_t \]
\[ w = \left( \sum_{t=1}^{T} E_t(x(s_t) - \gamma x(s_{t+1}))^T \right)^{-1} \sum_{t=1}^{T} E_t R_{t+1} \]
## Convergence of Linear Least Square Prediction

<table>
<thead>
<tr>
<th>On/Off policy</th>
<th>Algorithm</th>
<th>Table Lookup</th>
<th>Linear</th>
<th>Non-Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Policy</td>
<td>MC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>LSMC</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>TD</td>
<td>✔ ✔</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td></td>
<td>LSTD</td>
<td>✔ ✔</td>
<td>✔</td>
<td>-</td>
</tr>
<tr>
<td>Off-Policy</td>
<td>MC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>LSMC</td>
<td>✔ ✔ ✔</td>
<td>✔</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>TD</td>
<td>✔ ✔</td>
<td>✔</td>
<td>✘ ✘</td>
</tr>
<tr>
<td></td>
<td>Gradient TD</td>
<td>✔ ✔</td>
<td>✔</td>
<td>-</td>
</tr>
</tbody>
</table>
Least Squares Control Algorithms

• Policy evaluation: Policy evaluation by least squares Q-Learning
• Policy improvement: Greedy policy improvement
• We want to make use of all experience in $\mathcal{D}$ to:
  – Efficiency evaluate the policy
  – Improve the policy

But: The experience is drawn from different policies from various stages of training.
⇒ To evaluate $q_\pi(S, A)$ we must learn off-policy
⇒ Use the same idea as on Q-learning:
  • Use experience generated by old policy: $S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{old}$
  • Consider alternative successor action $A' = \pi_{new}(S_{t+1})$
  • Update $f(S_t, A_t, w)$ towards value of alternative action:

$$R_{t+1} + \gamma f(S_{t+1}, A', w)$$
Least Squares Q-Learning

- Given the following Q-learning update
  \[ \delta = R_{t+1} + \gamma f(S_{t+1}, \pi(S_{t+1}), w) - f(S_t, A_t, w) \]
  \[ \Delta w = \alpha \delta x(S_t, A_t) \]

- LSTDQ algorithm: solve for total update = zero
  \[ 0 = \sum_{t=1}^{T} \alpha (R_{t+1} + \gamma f(S_{t+1}, \pi(S_{t+1}), w) - f(S_t, A_t, w)) - f(S_t, A_t, w)x(S_t, A_t) \]
  \[ w = \left( \sum_{t=1}^{T} x(S_t, A_t) \left( x(S_t, A_t) - \gamma x(S_{t+1}, \pi(S_{t+1})) \right)^T \right)^{-1} \sum_{t=1}^{T} x(S_t, A_t) R_{t+1} \]
Least Squares Policy Iteration Algorithm

- Pseudocode Policy Iteration using LSTDQ
- Experience $\mathcal{D}$ is re-evaluated with different policies

\[\text{function } \text{LSPI-}TD(\mathcal{D}, \pi_0)\]
\[\pi' \leftarrow \pi_0\]
\[\text{repeat}\]
\[\pi \leftarrow \pi'\]
\[Q \leftarrow \text{LSTDQ}(\pi, \mathcal{D})\]
\[\text{for all } s \in S \text{ do}\]
\[\pi'(s) \leftarrow \arg\max_{a \in A} Q(s, a)\]
\[\text{end for}\]
\[\text{until } (\pi \approx \pi')\]
\[\text{return } \pi\]
End function
## Convergence of Control Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Table Lookup</th>
<th>Linear</th>
<th>Non-Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC Control</td>
<td>✔</td>
<td>(✔)</td>
<td>✘</td>
</tr>
<tr>
<td>Sarsa</td>
<td>✔</td>
<td>(✔)</td>
<td>✘</td>
</tr>
<tr>
<td>Q-Learning</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>LSPI</td>
<td>✔</td>
<td>(✔)</td>
<td>-</td>
</tr>
</tbody>
</table>

(✔) = jumps around the near optimal value function
Literature

- Lecture notes D. Silver: Introduction to Reinforcement Learning (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)

