A Guided Tour of Chapter 11:
Batch RL: Experience Replay, DQN, LSPI

Ashwin Rao

ICME, Stanford University

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data
- Let's understand the difference for the simple case of MC Prediction

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data
- Let's understand the difference for the simple case of MC Prediction
- Given fixed finite sequence of trace experiences yielding training data:

$$\mathcal{D} = [(S_i, G_i) | 1 \leq i \leq n]$$

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data
- Let's understand the difference for the simple case of MC Prediction
- Given fixed finite sequence of trace experiences yielding training data:

$$\mathcal{D} = [(S_i, G_i) | 1 \leq i \leq n]$$

- Incremental MC estimates $V(s; \boldsymbol{w})$ using $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ for each data pair:

$$\mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$

$$\nabla_{\boldsymbol{w}} \mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = (V(S_i; \boldsymbol{w}) - G_i) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

$$\Delta \boldsymbol{w} = \alpha \cdot (G_i - V(S_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data
- Let's understand the difference for the simple case of MC Prediction
- Given fixed finite sequence of trace experiences yielding training data:

$$\mathcal{D} = [(S_i, G_i) | 1 \leq i \leq n]$$

- Incremental MC estimates $V(s; \boldsymbol{w})$ using $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ for each data pair:

$$\mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$

$$\nabla_{\boldsymbol{w}} \mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = (V(S_i; \boldsymbol{w}) - G_i) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

$$\Delta \boldsymbol{w} = \alpha \cdot (G_i - V(S_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

- n updates are performed in sequence for $i = 1, 2, \ldots, n$

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data
- Let's understand the difference for the simple case of MC Prediction
- Given fixed finite sequence of trace experiences yielding training data:

$$\mathcal{D} = [(S_i, G_i) | 1 \leq i \leq n]$$

- Incremental MC estimates $V(s; \boldsymbol{w})$ using $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ for each data pair:

$$\mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$

$$\nabla_{\boldsymbol{w}} \mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = (V(S_i; \boldsymbol{w}) - G_i) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

$$\Delta \boldsymbol{w} = \alpha \cdot (G_i - V(S_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

- n updates are performed in sequence for $i = 1, 2, \ldots, n$
- Uses update method of FunctionApprox for each data pair (S_i, G_i)

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data
- Let's understand the difference for the simple case of MC Prediction
- Given fixed finite sequence of trace experiences yielding training data:

$$\mathcal{D} = [(S_i, G_i) | 1 \leq i \leq n]$$

- Incremental MC estimates $V(s; \boldsymbol{w})$ using $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ for each data pair:

$$\mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$

$$\nabla_{\boldsymbol{w}} \mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = (V(S_i; \boldsymbol{w}) - G_i) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

$$\Delta \boldsymbol{w} = \alpha \cdot (G_i - V(S_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

- n updates are performed in sequence for $i = 1, 2, \ldots, n$
- Uses update method of FunctionApprox for each data pair (S_i, G_i)
- Incremental RL makes inefficient use of available training data $\mathcal{D}$

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data
- Let's understand the difference for the simple case of MC Prediction
- Given fixed finite sequence of trace experiences yielding training data:

$$\mathcal{D} = [(S_i, G_i) | 1 \leq i \leq n]$$

- Incremental MC estimates $V(s; \boldsymbol{w})$ using $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ for each data pair:

$$\mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$

$$\nabla_{\boldsymbol{w}} \mathcal{L}_{(S_i, G_i)}(\boldsymbol{w}) = (V(S_i; \boldsymbol{w}) - G_i) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

$$\Delta \boldsymbol{w} = \alpha \cdot (G_i - V(S_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S_i; \boldsymbol{w})$$

- n updates are performed in sequence for $i = 1, 2, \ldots, n$
- Uses update method of FunctionApprox for each data pair (S_i, G_i)
- Incremental RL makes inefficient use of available training data $\mathcal{D}$
- Essentially each data point is "discarded" after being used for update

Batch MC Prediction makes efficient use of training data

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \boldsymbol{w}^*)$ such that

$$\begin{aligned}\boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2 \\ &= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]\end{aligned}$$

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \boldsymbol{w}^*)$ such that

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$
$$= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]$$

- This is the solve method of FunctionApprox on training data $\mathcal{D}$

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \boldsymbol{w}^*)$ such that

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$
$$= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]$$

- This is the solve method of FunctionApprox on training data $\mathcal{D}$
- This approach to RL is known as *Batch RL*

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \boldsymbol{w}^*)$ such that

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$
$$= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]$$

- This is the `solve` method of `FunctionApprox` on training data $\mathcal{D}$
- This approach to RL is known as *Batch RL*
- `solve` by doing updates with repeated use of available data pairs

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \boldsymbol{w}^*)$ such that

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$
$$= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]$$

- This is the `solve` method of `FunctionApprox` on training data $\mathcal{D}$
- This approach to RL is known as *Batch RL*
- `solve` by doing `update`s with repeated use of available data pairs
- Each update using random data pair $(S, G) \sim \mathcal{D}$

$$\Delta \boldsymbol{w} = \alpha \cdot (G - V(S; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S; \boldsymbol{w})$$

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \boldsymbol{w}^*)$ such that

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$
$$= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]$$

- This is the `solve` method of `FunctionApprox` on training data $\mathcal{D}$
- This approach to RL is known as *Batch RL*
- `solve` by doing updates with repeated use of available data pairs
- Each update using random data pair $(S, G) \sim \mathcal{D}$

$$\Delta \boldsymbol{w} = \alpha \cdot (G - V(S; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S; \boldsymbol{w})$$

- This will ultimately converge to desired value function $V(s; \boldsymbol{w}^*)$

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \boldsymbol{w}^*)$ such that

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$
$$= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]$$

- This is the `solve` method of `FunctionApprox` on training data $\mathcal{D}$
- This approach to RL is known as *Batch RL*
- `solve` by doing updates with repeated use of available data pairs
- Each update using random data pair $(S, G) \sim \mathcal{D}$

$$\Delta \boldsymbol{w} = \alpha \cdot (G - V(S; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S; \boldsymbol{w})$$

- This will ultimately converge to desired value function $V(s; \boldsymbol{w}^*)$
- Repeated use of available data known as *Experience Replay*

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \boldsymbol{w}^*)$ such that

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$
$$= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]$$

- This is the `solve` method of `FunctionApprox` on training data $\mathcal{D}$
- This approach to RL is known as *Batch RL*
- `solve` by doing updates with repeated use of available data pairs
- Each update using random data pair $(S, G) \sim \mathcal{D}$

$$\Delta \boldsymbol{w} = \alpha \cdot (G - V(S; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S; \boldsymbol{w})$$

- This will ultimately converge to desired value function $V(s; \boldsymbol{w}^*)$
- Repeated use of available data known as *Experience Replay*
- This makes more efficient use of available training data $\mathcal{D}$

Batch TD Prediction makes efficient use of *Experience*

Batch TD Prediction makes efficient use of *Experience*

- In Batch TD Prediction, we have experience $\mathcal{D}$ available as:

$$\mathcal{D} = [(S_i, R_i, S'_i)|1 \leq i \leq n]$$

Batch TD Prediction makes efficient use of *Experience*

- In Batch TD Prediction, we have experience $\mathcal{D}$ available as:

$$\mathcal{D} = [(S_i, R_i, S'_i) | 1 \leq i \leq n]$$

- Where (R_i, S'_i) is the pair of reward and next state from a state S_i

Batch TD Prediction makes efficient use of *Experience*

- In Batch TD Prediction, we have experience $\mathcal{D}$ available as:

$$\mathcal{D} = [(S_i, R_i, S'_i) | 1 \leq i \leq n]$$

- Where (R_i, S'_i) is the pair of reward and next state from a state S_i
- So, Experience $\mathcal{D}$ in the form of finite number of atomic experiences

Batch TD Prediction makes efficient use of *Experience*

- In Batch TD Prediction, we have experience $\mathcal{D}$ available as:

$$\mathcal{D} = [(S_i, R_i, S'_i) | 1 \leq i \leq n]$$

- Where (R_i, S'_i) is the pair of reward and next state from a state S_i
- So, Experience $\mathcal{D}$ in the form of finite number of atomic experiences
- This is represented in code as an Iterable[TransitionStep[S]]

Batch TD Prediction makes efficient use of *Experience*

- In Batch TD Prediction, we have experience $\mathcal{D}$ available as:

$$\mathcal{D} = [(S_i, R_i, S'_i) | 1 \leq i \leq n]$$

- Where (R_i, S'_i) is the pair of reward and next state from a state S_i
- So, Experience $\mathcal{D}$ in the form of finite number of atomic experiences
- This is represented in code as an $\text{Iterable}[\text{TransitionStep}[S]]$
- Parameters updated with repeated use of these atomic experiences

Batch TD Prediction makes efficient use of *Experience*

- In Batch TD Prediction, we have experience $\mathcal{D}$ available as:

$$\mathcal{D} = [(S_i, R_i, S'_i) | 1 \leq i \leq n]$$

- Where (R_i, S'_i) is the pair of reward and next state from a state S_i
- So, Experience $\mathcal{D}$ in the form of finite number of atomic experiences
- This is represented in code as an $\text{Iterable}[\text{TransitionStep}[\text{S}]]$
- Parameters updated with repeated use of these atomic experiences
- Each update using random data pair $(S, R, S') \sim \mathcal{D}$

$$\Delta \boldsymbol{w} = \alpha \cdot (R + \gamma \cdot V(S'; \boldsymbol{w}) - V(S; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S; \boldsymbol{w})$$

Batch TD Prediction makes efficient use of *Experience*

- In Batch TD Prediction, we have experience $\mathcal{D}$ available as:

$$\mathcal{D} = [(S_i, R_i, S'_i) | 1 \leq i \leq n]$$

- Where (R_i, S'_i) is the pair of reward and next state from a state S_i
- So, Experience $\mathcal{D}$ in the form of finite number of atomic experiences
- This is represented in code as an `Iterable[TransitionStep[S]]`
- Parameters updated with repeated use of these atomic experiences
- Each update using random data pair $(S, R, S') \sim \mathcal{D}$

$$\Delta \boldsymbol{w} = \alpha \cdot (R + \gamma \cdot V(S'; \boldsymbol{w}) - V(S; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} V(S; \boldsymbol{w})$$

- This is TD Prediction with Experience Replay on Finite Experience $\mathcal{D}$

Batch TD(λ) Prediction

Batch TD(λ) Prediction

- In Batch TD(λ) Prediction, given finite number of trace experiences

$$\mathcal{D} = [(S_{i,0}, R_{i,1}, S_{i,1}, R_{i,2}, S_{i,2}, \ldots, R_{i,T_i}, S_{i,T_i}) | 1 \leq i \leq n]$$

Batch TD(λ) Prediction

- In Batch TD(λ) Prediction, given finite number of trace experiences

$$\mathcal{D} = [(S_{i,0}, R_{i,1}, S_{i,1}, R_{i,2}, S_{i,2}, \ldots, R_{i,T_i}, S_{i,T_i}) | 1 \leq i \leq n]$$

- Parameters updated with repeated use of these trace experiences

Batch TD(λ) Prediction

- In Batch TD(λ) Prediction, given finite number of trace experiences

$$\mathcal{D} = [(S_{i,0}, R_{i,1}, S_{i,1}, R_{i,2}, S_{i,2}, \ldots, R_{i,T_i}, S_{i,T_i}) | 1 \leq i \leq n]$$

- Parameters updated with repeated use of these trace experiences
- Randomly pick trace experience (say indexed i) $\sim \mathcal{D}$

Batch TD(λ) Prediction

- In Batch TD(λ) Prediction, given finite number of trace experiences

$$\mathcal{D} = [(S_{i,0}, R_{i,1}, S_{i,1}, R_{i,2}, S_{i,2}, \ldots, R_{i,T_i}, S_{i,T_i}) | 1 \leq i \leq n]$$

- Parameters updated with repeated use of these trace experiences
- Randomly pick trace experience (say indexed i) $\sim \mathcal{D}$
- For trace experience i, parameters updated at each time step t:

$$\boldsymbol{E}_t = \gamma\lambda \cdot \boldsymbol{E}_{t-1} + \nabla_{\boldsymbol{w}} V(S_{i,t}; \boldsymbol{w})$$

$$\Delta \boldsymbol{w} = \alpha \cdot (R_{i,t+1} + \gamma \cdot V(S_{i,t+1}; \boldsymbol{w}) - V(S_{i,t}; \boldsymbol{w})) \cdot \boldsymbol{E}_t$$

The Deep Q-Networks (DQN) Control Algorithm

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; \boldsymbol{w})$

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; \boldsymbol{w})$
- Given state S_t and action A_t, obtain reward R_{t+1} and next state S_{t+1}

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; \boldsymbol{w})$
- Given state S_t and action A_t, obtain reward R_{t+1} and next state S_{t+1}
- Store atomic experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory $\mathcal{D}$

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; \boldsymbol{w})$
- Given state S_t and action A_t, obtain reward R_{t+1} and next state S_{t+1}
- Store atomic experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of atomic experiences $(s_i, a_i, r_i, s'_i) \sim \mathcal{D}$

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; \boldsymbol{w})$
- Given state S_t and action A_t, obtain reward R_{t+1} and next state S_{t+1}
- Store atomic experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of atomic experiences $(s_i, a_i, r_i, s'_i) \sim \mathcal{D}$
- Update Q-network parameters $\boldsymbol{w}$ using Q-learning targets based on "frozen" parameters $\boldsymbol{w}^-$ of *target network*

$$\Delta \boldsymbol{w} = \alpha \cdot \sum_i (r_i + \gamma \cdot \max_{a'_i} Q(s'_i, a'_i; \boldsymbol{w}^-) - Q(s_i, a_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(s_i, a_i; \boldsymbol{w})$$

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; \boldsymbol{w})$
- Given state S_t and action A_t, obtain reward R_{t+1} and next state S_{t+1}
- Store atomic experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of atomic experiences $(s_i, a_i, r_i, s'_i) \sim \mathcal{D}$
- Update Q-network parameters $\boldsymbol{w}$ using Q-learning targets based on "frozen" parameters $\boldsymbol{w}^-$ of *target network*

$$\Delta \boldsymbol{w} = \alpha \cdot \sum_i (r_i + \gamma \cdot \max_{a'_i} Q(s'_i, a'_i; \boldsymbol{w}^-) - Q(s_i, a_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(s_i, a_i; \boldsymbol{w})$$

- $S_t \leftarrow S_{t+1}$

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; \boldsymbol{w})$
- Given state S_t and action A_t, obtain reward R_{t+1} and next state S_{t+1}
- Store atomic experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of atomic experiences $(s_i, a_i, r_i, s'_i) \sim \mathcal{D}$
- Update Q-network parameters $\boldsymbol{w}$ using Q-learning targets based on "frozen" parameters $\boldsymbol{w}^-$ of *target network*

$$\Delta \boldsymbol{w} = \alpha \cdot \sum_i (r_i + \gamma \cdot \max_{a'_i} Q(s'_i, a'_i; \boldsymbol{w}^-) - Q(s_i, a_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(s_i, a_i; \boldsymbol{w})$$

- $S_t \leftarrow S_{t+1}$

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; \boldsymbol{w})$
- Given state S_t and action A_t, obtain reward R_{t+1} and next state S_{t+1}
- Store atomic experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of atomic experiences $(s_i, a_i, r_i, s'_i) \sim \mathcal{D}$
- Update Q-network parameters $\boldsymbol{w}$ using Q-learning targets based on "frozen" parameters $\boldsymbol{w}^-$ of *target network*

$$\Delta \boldsymbol{w} = \alpha \cdot \sum_i (r_i + \gamma \cdot \max_{a'_i} Q(s'_i, a'_i; \boldsymbol{w}^-) - Q(s_i, a_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(s_i, a_i; \boldsymbol{w})$$

- $S_t \leftarrow S_{t+1}$

Parameters $\boldsymbol{w}^-$ of target network infrequently updated to values of Q-network parameters $\boldsymbol{w}$ (hence, Q-learning targets treated as "frozen")

Least-Squares RL Prediction

Least-Squares RL Prediction

- Batch RL Prediction for general function approximation is iterative

Least-Squares RL Prediction

- Batch RL Prediction for general function approximation is iterative
- Uses experience replay and gradient descent

Least-Squares RL Prediction

- Batch RL Prediction for general function approximation is iterative
- Uses experience replay and gradient descent
- We can solve directly (without gradient) for linear function approx

Least-Squares RL Prediction

- Batch RL Prediction for general function approximation is iterative
- Uses experience replay and gradient descent
- We can solve directly (without gradient) for linear function approx
- Define a sequence of feature functions $\phi_j : \mathcal{X} \rightarrow \mathbb{R}, j = 1, 2, \ldots, m$

Least-Squares RL Prediction

- Batch RL Prediction for general function approximation is iterative
- Uses experience replay and gradient descent
- We can solve directly (without gradient) for linear function approx
- Define a sequence of feature functions $\phi_j : \mathcal{X} \rightarrow \mathbb{R}, j = 1, 2, \ldots, m$
- Parameters w is a weights vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_m) \in \mathbb{R}^m$

Least-Squares RL Prediction

- Batch RL Prediction for general function approximation is iterative
- Uses experience replay and gradient descent
- We can solve directly (without gradient) for linear function approx
- Define a sequence of feature functions $\phi_j : \mathcal{X} \rightarrow \mathbb{R}, j = 1, 2, \ldots, m$
- Parameters w is a weights vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_m) \in \mathbb{R}^m$
- Value Function is approximated as:

$$V(s; \boldsymbol{w}) = \sum_{j=1}^{m} \phi_j(s) \cdot w_j = \boldsymbol{\phi}(s)^T \cdot \boldsymbol{w}$$

where $\boldsymbol{\phi}(s) \in \mathbb{R}^m$ is the feature vector for state s

Least-Squares Monte-Carlo (LSMC)

Least-Squares Monte-Carlo (LSMC)

- Loss function for Batch MC Prediction with data $[(S_i, G_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\sum_{j=1}^{m} \phi_j(S_i) \cdot w_j - G_i)^2 = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w} - G_i)^2$$

Least-Squares Monte-Carlo (LSMC)

- Loss function for Batch MC Prediction with data $[(S_i, G_i)|1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\sum_{j=1}^{m} \phi_j(S_i) \cdot w_j - G_i)^2 = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w} - G_i)^2$$

- The gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \boldsymbol{\phi}(S_i) \cdot (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w}^* - G_i) = 0$$

Least-Squares Monte-Carlo (LSMC)

- Loss function for Batch MC Prediction with data $[(S_i, G_i)|1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\sum_{j=1}^{m} \phi_j(S_i) \cdot w_j - G_i)^2 = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w} - G_i)^2$$

- The gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \boldsymbol{\phi}(S_i) \cdot (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w^*} - G_i) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$

Least-Squares Monte-Carlo (LSMC)

- Loss function for Batch MC Prediction with data $[(S_i, G_i)|1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\sum_{j=1}^{m} \phi_j(S_i) \cdot w_j - G_i)^2 = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w} - G_i)^2$$

- The gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \boldsymbol{\phi}(S_i) \cdot (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w^*} - G_i) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each data pair (S_i, G_i) as:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{\phi}(S_i) \cdot \boldsymbol{\phi}(S_i)^T \text{ (i.e., outer-product of } \boldsymbol{\phi}(S_i) \text{ with itself)}$$

Least-Squares Monte-Carlo (LSMC)

- Loss function for Batch MC Prediction with data $[(S_i, G_i)|1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\sum_{j=1}^{m} \phi_j(S_i) \cdot w_j - G_i)^2 = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w} - G_i)^2$$

- The gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \boldsymbol{\phi}(S_i) \cdot (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w^*} - G_i) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each data pair (S_i, G_i) as:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{\phi}(S_i) \cdot \boldsymbol{\phi}(S_i)^T \text{ (i.e., outer-product of } \boldsymbol{\phi}(S_i) \text{ with itself)}$$

- m-Vector $\boldsymbol{b}$ is accumulated at each data pair (S_i, G_i) as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{\phi}(S_i) \cdot G_i$$

Least-Squares Monte-Carlo (LSMC)

- Loss function for Batch MC Prediction with data $[(S_i, G_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\sum_{j=1}^{m} \phi_j(S_i) \cdot w_j - G_i)^2 = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w} - G_i)^2$$

- The gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \boldsymbol{\phi}(S_i) \cdot (\boldsymbol{\phi}(S_i)^T \cdot \boldsymbol{w^*} - G_i) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each data pair (S_i, G_i) as:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{\phi}(S_i) \cdot \boldsymbol{\phi}(S_i)^T \text{ (i.e., outer-product of } \boldsymbol{\phi}(S_i) \text{ with itself)}$$

- m-Vector $\boldsymbol{b}$ is accumulated at each data pair (S_i, G_i) as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{\phi}(S_i) \cdot G_i$$

- Shermann-Morrison incremental inverse can be done in $O(m^2)$

Least-Squares Temporal-Difference (LSTD)

Least-Squares Temporal-Difference (LSTD)

- Loss func for Batch TD Prediction with data $[(s_i, r_i, s'_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\phi(s_i) \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}))^2$$

Least-Squares Temporal-Difference (LSTD)

- Loss func for Batch TD Prediction with data $[(s_i, r_i, s'_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\phi(s_i) \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}))^2$$

- The semi-gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^*)) = 0$$

Least-Squares Temporal-Difference (LSTD)

- Loss func for Batch TD Prediction with data $[(s_i, r_i, s'_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\phi(s_i) \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}))^2$$

- The semi-gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w^*} - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^*)) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$

Least-Squares Temporal-Difference (LSTD)

- Loss func for Batch TD Prediction with data $[(s_i, r_i, s'_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\phi(s_i) \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}))^2$$

- The semi-gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^*)) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience (s_i, r_i, s'_i):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i) \cdot (\phi(s_i) - \gamma \cdot \phi(s'_i))^T \text{ (note the Outer-Product)}$$

Least-Squares Temporal-Difference (LSTD)

- Loss func for Batch TD Prediction with data $[(s_i, r_i, s'_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\phi(s_i) \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}))^2$$

- The semi-gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^*)) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience (s_i, r_i, s'_i):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i) \cdot (\phi(s_i) - \gamma \cdot \phi(s'_i))^T \text{ (note the Outer-Product)}$$

- m-Vector $\boldsymbol{b}$ is accumulated at each atomic experience (s_i, r_i, s'_i):

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \phi(s_i) \cdot r_i$$

Least-Squares Temporal-Difference (LSTD)

- Loss func for Batch TD Prediction with data $[(s_i, r_i, s'_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\phi(s_i) \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}))^2$$

- The semi-gradient of this Loss function is set to 0 to solve for $\boldsymbol{w}^*$

$$\sum_{i=1}^{n} \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^*)) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience (s_i, r_i, s'_i):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i) \cdot (\phi(s_i) - \gamma \cdot \phi(s'_i))^T \text{ (note the Outer-Product)}$$

- m-Vector $\boldsymbol{b}$ is accumulated at each atomic experience (s_i, r_i, s'_i):

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \phi(s_i) \cdot r_i$$

- Shermann-Morrison incremental inverse can be done in $O(m^2)$

LSTD(λ)

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces
- Denote the Eligibility Trace of atomic experience i as $\boldsymbol{E}_i$

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces
- Denote the Eligibility Trace of atomic experience i as $\boldsymbol{E}_i$
- Note: $\boldsymbol{E}_i$ accumulates $\nabla_{\boldsymbol{w}} V(s; \boldsymbol{w}) = \phi(s)$ in each trace experience

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces
- Denote the Eligibility Trace of atomic experience i as $\boldsymbol{E}_i$
- Note: $\boldsymbol{E}_i$ accumulates $\nabla_{\boldsymbol{w}} V(s; \boldsymbol{w}) = \phi(s)$ in each trace experience
- When accumulating, previous step's eligibility trace discounted by $\lambda\gamma$

$$\sum_{i=1}^{n} \boldsymbol{E_i} \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s_i')^T \cdot \boldsymbol{w}^*)) = 0$$

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces
- Denote the Eligibility Trace of atomic experience i as $\boldsymbol{E_i}$
- Note: $\boldsymbol{E_i}$ accumulates $\nabla_{\boldsymbol{w}} V(s; \boldsymbol{w}) = \phi(s)$ in each trace experience
- When accumulating, previous step's eligibility trace discounted by $\lambda\gamma$

$$\sum_{i=1}^{n} \boldsymbol{E_i} \cdot (\phi(s_i)^T \cdot \boldsymbol{w^*} - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^*)) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces
- Denote the Eligibility Trace of atomic experience i as $\boldsymbol{E_i}$
- Note: $\boldsymbol{E_i}$ accumulates $\nabla_{\boldsymbol{w}} V(s; \boldsymbol{w}) = \phi(s)$ in each trace experience
- When accumulating, previous step's eligibility trace discounted by $\lambda\gamma$

$$\sum_{i=1}^{n} \boldsymbol{E_i} \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^*)) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience i:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{E_i} \cdot (\phi(s_i) - \gamma \cdot \phi(s'_i))^T \text{ (note the Outer-Product)}$$

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces
- Denote the Eligibility Trace of atomic experience i as $\boldsymbol{E}_i$
- Note: $\boldsymbol{E}_i$ accumulates $\nabla_{\boldsymbol{w}} V(s;\boldsymbol{w}) = \phi(s)$ in each trace experience
- When accumulating, previous step's eligibility trace discounted by $\lambda\gamma$

$$\sum_{i=1}^{n} \boldsymbol{E_i} \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^*)) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience i:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{E_i} \cdot (\phi(s_i) - \gamma \cdot \phi(s'_i))^T \text{ (note the Outer-Product)}$$

- m-Vector $\boldsymbol{b}$ is accumulated at each atomic experience (s_i, r_i, s'_i) as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{E_i} \cdot r_i$$

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces
- Denote the Eligibility Trace of atomic experience i as $\boldsymbol{E}_i$
- Note: $\boldsymbol{E}_i$ accumulates $\nabla_{\boldsymbol{w}} V(s; \boldsymbol{w}) = \phi(s)$ in each trace experience
- When accumulating, previous step's eligibility trace discounted by $\lambda\gamma$

$$\sum_{i=1}^{n} \boldsymbol{E_i} \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s_i')^T \cdot \boldsymbol{w}^*)) = 0$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience i:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{E_i} \cdot (\phi(s_i) - \gamma \cdot \phi(s_i'))^T \text{ (note the Outer-Product)}$$

- m-Vector $\boldsymbol{b}$ is accumulated at each atomic experience (s_i, r_i, s_i') as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{E_i} \cdot r_i$$

- Shermann-Morrison incremental inverse can be done in $O(m^2)$

Convergence of Least Squares Prediction Algorithms

Convergence of Least Squares Prediction Algorithms

On/Off Policy	Algorithm	Tabular	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✗	-
	TD	✓	✗	✗
	LSTD	✓	✗	-

Least Squares RL Control

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:
  - Policy Evaluation as Least-Squares Q-Value Prediction

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:
    - Policy Evaluation as Least-Squares Q-Value Prediction
    - Greedy Policy Improvement

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:
  - Policy Evaluation as Least-Squares Q-Value Prediction
  - Greedy Policy Improvement
- For MC or On-Policy TD Control, Q-Value Prediction (for policy π):

$$Q^\pi(s,a) \approx Q(s,a;\boldsymbol{w}^*) = \phi(s,a)^T \cdot \boldsymbol{w}^*$$

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:
  - Policy Evaluation as Least-Squares Q-Value Prediction
  - Greedy Policy Improvement
- For MC or On-Policy TD Control, Q-Value Prediction (for policy π):

$$Q^\pi(s,a) \approx Q(s,a;\boldsymbol{w}^*) = \phi(s,a)^T \cdot \boldsymbol{w}^*$$

- Direct solve for $\boldsymbol{w}^*$ using experience data generated using policy π

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:
  - Policy Evaluation as Least-Squares Q-Value Prediction
  - Greedy Policy Improvement
- For MC or On-Policy TD Control, Q-Value Prediction (for policy π):

$$Q^\pi(s,a) \approx Q(s,a;\boldsymbol{w}^*) = \phi(s,a)^T \cdot \boldsymbol{w}^*$$

- Direct solve for $\boldsymbol{w}^*$ using experience data generated using policy π
- We are interested in Off-Policy Control with Least-Squares TD

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:
  - Policy Evaluation as Least-Squares Q-Value Prediction
  - Greedy Policy Improvement
- For MC or On-Policy TD Control, Q-Value Prediction (for policy π):

$$Q^{\pi}(s,a) \approx Q(s,a;\boldsymbol{w}^*) = \phi(s,a)^T \cdot \boldsymbol{w}^*$$

- Direct solve for $\boldsymbol{w}^*$ using experience data generated using policy π
- We are interested in Off-Policy Control with Least-Squares TD
- Using the same idea as Q-Learning and with Experience Replay

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:
  - Policy Evaluation as Least-Squares Q-Value Prediction
  - Greedy Policy Improvement
- For MC or On-Policy TD Control, Q-Value Prediction (for policy π):

$$Q^\pi(s,a) \approx Q(s,a;\boldsymbol{w}^*) = \phi(s,a)^T \cdot \boldsymbol{w}^*$$

- Direct solve for $\boldsymbol{w}^*$ using experience data generated using policy π
- We are interested in Off-Policy Control with Least-Squares TD
- Using the same idea as Q-Learning and with Experience Replay
- This technique is known as Least Squares Policy Iteration (LSPI)

Least Squares Policy Iteration (LSPI)

Least Squares Policy Iteration (LSPI)

- Each iteration of GPI starts with a function approximation

$$Q(s, a; \boldsymbol{w}) = \phi(s, a)^T \cdot \boldsymbol{w} = \sum_{j=1}^{m} \phi_j(s, a) \cdot w_j$$

Least Squares Policy Iteration (LSPI)

- Each iteration of GPI starts with a function approximation

$$Q(s, a; \boldsymbol{w}) = \phi(s, a)^T \cdot \boldsymbol{w} = \sum_{j=1}^{m} \phi_j(s, a) \cdot w_j$$

- Deterministic policy π_D (*target policy* for this iteration) is given by:

$$\pi_D(s) = \arg\max_a Q(s, a; \boldsymbol{w})$$

Least Squares Policy Iteration (LSPI)

- Each iteration of GPI starts with a function approximation

$$Q(s, a; \boldsymbol{w}) = \phi(s, a)^T \cdot \boldsymbol{w} = \sum_{j=1}^{m} \phi_j(s, a) \cdot w_j$$

- Deterministic policy π_D (*target policy* for this iteration) is given by:

$$\pi_D(s) = \arg\max_a Q(s, a; \boldsymbol{w})$$

- Sample mini-batch of experiences (s_i, a_i, r_i, s'_i) from replay memory $\mathcal{D}$

Least Squares Policy Iteration (LSPI)

- Each iteration of GPI starts with a function approximation

$$Q(s, a; \boldsymbol{w}) = \phi(s, a)^T \cdot \boldsymbol{w} = \sum_{j=1}^{m} \phi_j(s, a) \cdot w_j$$

- Deterministic policy π_D (*target policy* for this iteration) is given by:

$$\pi_D(s) = \arg\max_a Q(s, a; \boldsymbol{w})$$

- Sample mini-batch of experiences (s_i, a_i, r_i, s'_i) from replay memory $\mathcal{D}$
- Goal of the iteration is to solve for weights $\boldsymbol{w}^*$ to minimize:

$$\mathcal{L}(\boldsymbol{w}) = \sum_i (Q(s_i, a_i; \boldsymbol{w}) - (r_i + \gamma \cdot Q(s'_i, \pi_D(s'_i); \boldsymbol{w})))^2$$
$$= \sum_i (\phi(s_i, a_i)^T \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i))^T \cdot \boldsymbol{w}))^2$$

Least Squares Policy Iteration (LSPI)

- Each iteration of GPI starts with a function approximation

$$Q(s, a; \boldsymbol{w}) = \phi(s, a)^T \cdot \boldsymbol{w} = \sum_{j=1}^{m} \phi_j(s, a) \cdot w_j$$

- Deterministic policy π_D (*target policy* for this iteration) is given by:

$$\pi_D(s) = \arg\max_a Q(s, a; \boldsymbol{w})$$

- Sample mini-batch of experiences (s_i, a_i, r_i, s'_i) from replay memory $\mathcal{D}$
- Goal of the iteration is to solve for weights $\boldsymbol{w}^*$ to minimize:

$$\mathcal{L}(\boldsymbol{w}) = \sum_i (Q(s_i, a_i; \boldsymbol{w}) - (r_i + \gamma \cdot Q(s'_i, \pi_D(s'_i); \boldsymbol{w})))^2$$
$$= \sum_i (\phi(s_i, a_i)^T \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i))^T \cdot \boldsymbol{w}))^2$$

- Iteration ends by setting the next iteration's start parameters $\boldsymbol{w}$ to $\boldsymbol{w}^*$

Least Squares Policy Iteration (LSPI)

Least Squares Policy Iteration (LSPI)

- We set the semi-gradient of $\mathcal{L}(\boldsymbol{w}^*)$ to 0

$$\sum_i \phi(s_i, a_i) \cdot (\phi(s_i, a_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i))^T \cdot \boldsymbol{w}^*)) = 0 \quad (1)$$

Least Squares Policy Iteration (LSPI)

- We set the semi-gradient of $\mathcal{L}(\boldsymbol{w}^*)$ to 0

$$\sum_i \phi(s_i, a_i) \cdot (\boldsymbol{\phi}(s_i, a_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \boldsymbol{\phi}(s'_i, \pi_D(s'_i))^T \cdot \boldsymbol{w}^*)) = 0 \quad (1)$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$

Least Squares Policy Iteration (LSPI)

- We set the semi-gradient of $\mathcal{L}(\boldsymbol{w}^*)$ to 0

$$\sum_i \phi(s_i, a_i) \cdot (\boldsymbol{\phi}(s_i, a_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \boldsymbol{\phi}(s'_i, \pi_D(s'_i))^T \cdot \boldsymbol{w}^*)) = 0 \quad (1)$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each experience (s_i, a_i, r_i, s'_i):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i, a_i) \cdot (\phi(s_i, a_i) - \gamma \cdot \phi(s'_i, \pi_D(s'_i)))^T$$

Least Squares Policy Iteration (LSPI)

- We set the semi-gradient of $\mathcal{L}(\boldsymbol{w}^*)$ to 0

$$\sum_i \phi(s_i, a_i) \cdot (\boldsymbol{\phi}(s_i, a_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \boldsymbol{\phi}(s'_i, \pi_D(s'_i))^T \cdot \boldsymbol{w}^*)) = 0 \quad (1)$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each experience (s_i, a_i, r_i, s'_i):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{\phi}(s_i, a_i) \cdot (\boldsymbol{\phi}(s_i, a_i) - \gamma \cdot \boldsymbol{\phi}(s'_i, \pi_D(s'_i)))^T$$

- m-Vector $\boldsymbol{b}$ is accumulated at each experience (s_i, a_i, r_i, s'_i) as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{\phi}(s_i, a_i) \cdot r_i$$

Least Squares Policy Iteration (LSPI)

- We set the semi-gradient of $\mathcal{L}(\boldsymbol{w}^*)$ to 0

$$\sum_i \phi(s_i, a_i) \cdot (\phi(s_i, a_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s_i', \pi_D(s_i'))^T \cdot \boldsymbol{w}^*)) = 0 \quad (1)$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each experience (s_i, a_i, r_i, s_i'):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i, a_i) \cdot (\phi(s_i, a_i) - \gamma \cdot \phi(s_i', \pi_D(s_i')))^T$$

- m-Vector $\boldsymbol{b}$ is accumulated at each experience (s_i, a_i, r_i, s_i') as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \phi(s_i, a_i) \cdot r_i$$

- Shermann-Morrison incremental inverse can be done in $O(m^2)$

Least Squares Policy Iteration (LSPI)

- We set the semi-gradient of $\mathcal{L}(\boldsymbol{w}^*)$ to 0

$$\sum_i \phi(s_i, a_i) \cdot (\phi(s_i, a_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i))^T \cdot \boldsymbol{w}^*)) = 0 \quad (1)$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each experience (s_i, a_i, r_i, s'_i):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i, a_i) \cdot (\phi(s_i, a_i) - \gamma \cdot \phi(s'_i, \pi_D(s'_i)))^T$$

- m-Vector $\boldsymbol{b}$ is accumulated at each experience (s_i, a_i, r_i, s'_i) as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \phi(s_i, a_i) \cdot r_i$$

- Shermann-Morrison incremental inverse can be done in $O(m^2)$
- This least-squares solution of $\boldsymbol{w}^*$ (Prediction) is known as *LSTDQ*

Least Squares Policy Iteration (LSPI)

- We set the semi-gradient of $\mathcal{L}(\boldsymbol{w}^*)$ to 0

$$\sum_i \phi(s_i, a_i) \cdot (\phi(s_i, a_i)^T \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i))^T \cdot \boldsymbol{w}^*)) = 0 \quad (1)$$

- $\boldsymbol{w}^*$ is solved as $\boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each experience (s_i, a_i, r_i, s'_i):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i, a_i) \cdot (\phi(s_i, a_i) - \gamma \cdot \phi(s'_i, \pi_D(s'_i)))^T$$

- m-Vector $\boldsymbol{b}$ is accumulated at each experience (s_i, a_i, r_i, s'_i) as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \phi(s_i, a_i) \cdot r_i$$

- Shermann-Morrison incremental inverse can be done in $O(m^2)$
- This least-squares solution of $\boldsymbol{w}^*$ (Prediction) is known as *LSTDQ*
- GPI with LSTDQ and greedy policy improvement known as *LSPI*

Convergence of Control Algorithms

Convergence of Control Algorithms

Algorithm	Tabular	Linear	Non-Linear
MC Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-Learning	✓	✗	✗
LSPI	✓	(✓)	-

Convergence of Control Algorithms

Algorithm	Tabular	Linear	Non-Linear
MC Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-Learning	✓	✗	✗
LSPI	✓	(✓)	-

(✓) means it chatters around near-optimal Value Function

LSPI for Optimal Exercise of American Options

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid
- Otherwise, the standard approach is Longstaff-Schwartz algorithm

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid
- Otherwise, the standard approach is Longstaff-Schwartz algorithm
- Longstaff-Schwartz algorithm combines 3 ideas:

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid
- Otherwise, the standard approach is Longstaff-Schwartz algorithm
- Longstaff-Schwartz algorithm combines 3 ideas:
    - Valuation based on Monte-Carlo simulation

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid
- Otherwise, the standard approach is Longstaff-Schwartz algorithm
- Longstaff-Schwartz algorithm combines 3 ideas:
  - Valuation based on Monte-Carlo simulation
  - Function approximation of continuation value for in-the-money states

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid
- Otherwise, the standard approach is Longstaff-Schwartz algorithm
- Longstaff-Schwartz algorithm combines 3 ideas:
  - Valuation based on Monte-Carlo simulation
  - Function approximation of continuation value for in-the-money states
  - Backward-recursive determination of early exercise states

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid
- Otherwise, the standard approach is Longstaff-Schwartz algorithm
- Longstaff-Schwartz algorithm combines 3 ideas:
  - Valuation based on Monte-Carlo simulation
  - Function approximation of continuation value for in-the-money states
  - Backward-recursive determination of early exercise states
- We consider LSPI as an alternative approach for American Pricing

LSPI as an alternative to Longstaff-Schwartz

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Stop) or Continue

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Stop) or Continue
- *Reward* always 0, except upon Exercise (= Payoff)

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Stop) or Continue
- *Reward* always 0, except upon Exercise (= Payoff)
- *State*-transitions based on Underlying Security's Risk-Neutral Process

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Stop) or Continue
- *Reward* always 0, except upon Exercise (= Payoff)
- *State*-transitions based on Underlying Security's Risk-Neutral Process
- Key is function approximation of state-conditioned continuation value

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Stop) or Continue
- *Reward* always 0, except upon Exercise (= Payoff)
- *State*-transitions based on Underlying Security's Risk-Neutral Process
- Key is function approximation of state-conditioned continuation value
- Continuation Value $\Rightarrow$ Optimal Stopping $\Rightarrow$ Option Price

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Stop) or Continue
- *Reward* always 0, except upon Exercise (= Payoff)
- *State*-transitions based on Underlying Security's Risk-Neutral Process
- Key is function approximation of state-conditioned continuation value
- Continuation Value $\Rightarrow$ Optimal Stopping $\Rightarrow$ Option Price
- We customize LSPI to Optimal Exercise of American Options

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Stop) or Continue
- *Reward* always 0, except upon Exercise (= Payoff)
- *State*-transitions based on Underlying Security's Risk-Neutral Process
- Key is function approximation of state-conditioned continuation value
- Continuation Value $\Rightarrow$ Optimal Stopping $\Rightarrow$ Option Price
- We customize LSPI to Optimal Exercise of American Options
- Based on this paper by Li, Szepesvari, Schuurmans

LSPI customized for American Options Pricing

LSPI customized for American Options Pricing

- 2 actions: $a = c$ (continue the option) and $a = e$ (exercise the option)

LSPI customized for American Options Pricing

- 2 actions: $a = c$ (continue the option) and $a = e$ (exercise the option)
- Create function approx representation for $Q(s, a)$ only for $a = c$ since we know option payoff $g(s)$ for $a = e$, i.e., $Q(s, a) = g(s)$

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} \phi(s)^T \cdot \boldsymbol{w} & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

for feature funcs $\phi(\cdot) = [\phi_i(\cdot) | i = 1, \ldots, m]$ of only state & not action

LSPI customized for American Options Pricing

- 2 actions: $a = c$ (continue the option) and $a = e$ (exercise the option)
- Create function approx representation for $Q(s, a)$ only for $a = c$ since we know option payoff $g(s)$ for $a = e$, i.e., $Q(s, a) = g(s)$

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} \phi(s)^T \cdot \boldsymbol{w} & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

  for feature funcs $\phi(\cdot) = [\phi_i(\cdot) | i = 1, \ldots, m]$ of only state & not action
- Each iteration starts with $\boldsymbol{w}$ defining $\hat{Q}$ (and it's greedy policy π_D), and ends by solving for $\boldsymbol{w}^*$ setting next iteration's $\boldsymbol{w}$ to $\boldsymbol{w}^*$

LSPI customized for American Options Pricing

- 2 actions: $a = c$ (continue the option) and $a = e$ (exercise the option)
- Create function approx representation for $Q(s, a)$ only for $a = c$ since we know option payoff $g(s)$ for $a = e$, i.e., $Q(s, a) = g(s)$

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} \phi(s)^T \cdot \boldsymbol{w} & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

  for feature funcs $\phi(\cdot) = [\phi_i(\cdot) | i = 1, \ldots, m]$ of only state & not action
- Each iteration starts with $\boldsymbol{w}$ defining $\hat{Q}$ (and it's greedy policy π_D), and ends by solving for $\boldsymbol{w}^*$ setting next iteration's $\boldsymbol{w}$ to $\boldsymbol{w}^*$
- Since we learn Q-Value function for only $a = c$, behavior policy μ generating experience data for training is a constant function $\mu(s) = c$

LSPI customized for American Options Pricing

- 2 actions: $a = c$ (continue the option) and $a = e$ (exercise the option)
- Create function approx representation for $Q(s, a)$ only for $a = c$ since we know option payoff $g(s)$ for $a = e$, i.e., $Q(s, a) = g(s)$

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} \phi(s)^T \cdot \boldsymbol{w} & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

  for feature funcs $\phi(\cdot) = [\phi_i(\cdot) | i = 1, \ldots, m]$ of only state & not action
- Each iteration starts with $\boldsymbol{w}$ defining $\hat{Q}$ (and it's greedy policy π_D), and ends by solving for $\boldsymbol{w}^*$ setting next iteration's $\boldsymbol{w}$ to $\boldsymbol{w}^*$
- Since we learn Q-Value function for only $a = c$, behavior policy μ generating experience data for training is a constant function $\mu(s) = c$
- Also, for American Options, the reward for $a = c$ is 0

LSPI customized for American Options Pricing

- 2 actions: $a = c$ (continue the option) and $a = e$ (exercise the option)
- Create function approx representation for $Q(s, a)$ only for $a = c$ since we know option payoff $g(s)$ for $a = e$, i.e., $Q(s, a) = g(s)$

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} \phi(s)^T \cdot \boldsymbol{w} & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

  for feature funcs $\phi(\cdot) = [\phi_i(\cdot) | i = 1, \ldots, m]$ of only state & not action
- Each iteration starts with $\boldsymbol{w}$ defining $\hat{Q}$ (and it's greedy policy π_D), and ends by solving for $\boldsymbol{w}^*$ setting next iteration's $\boldsymbol{w}$ to $\boldsymbol{w}^*$
- Since we learn Q-Value function for only $a = c$, behavior policy μ generating experience data for training is a constant function $\mu(s) = c$
- Also, for American Options, the reward for $a = c$ is 0
- So each atomic experiences for training is of the form $(s, c, 0, s')$

LSPI customized for American Options Pricing

- 2 actions: $a = c$ (continue the option) and $a = e$ (exercise the option)
- Create function approx representation for $Q(s, a)$ only for $a = c$ since we know option payoff $g(s)$ for $a = e$, i.e., $Q(s, a) = g(s)$

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} \phi(s)^T \cdot \boldsymbol{w} & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

  for feature funcs $\phi(\cdot) = [\phi_i(\cdot) | i = 1, \ldots, m]$ of only state & not action
- Each iteration starts with $\boldsymbol{w}$ defining $\hat{Q}$ (and it's greedy policy π_D), and ends by solving for $\boldsymbol{w}^*$ setting next iteration's $\boldsymbol{w}$ to $\boldsymbol{w}^*$
- Since we learn Q-Value function for only $a = c$, behavior policy μ generating experience data for training is a constant function $\mu(s) = c$
- Also, for American Options, the reward for $a = c$ is 0
- So each atomic experiences for training is of the form $(s, c, 0, s')$
- So we represent each atomic experience for training as a 2-tuple (s, s')

LSPI customized for American Options Pricing

LSPI customized for American Options Pricing

- This reduces LSPI Semi-Gradient Equation (1) to:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - \gamma \cdot \hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)) = 0 \qquad (2)$$

LSPI customized for American Options Pricing

- This reduces LSPI Semi-Gradient Equation (1) to:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - \gamma \cdot \hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)) = 0 \tag{2}$$

- We need to consider two cases for the term $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$

LSPI customized for American Options Pricing

- This reduces LSPI Semi-Gradient Equation (1) to:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - \gamma \cdot \hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)) = 0 \tag{2}$$

- We need to consider two cases for the term $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$
  - $C1$: If s'_i is non-terminal and $\pi_D(s'_i) = c$ (i.e., $\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)$): Substitute $\phi(s'_i)^T \cdot \boldsymbol{w}^*$ for $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$ in Equation (2)

LSPI customized for American Options Pricing

- This reduces LSPI Semi-Gradient Equation (1) to:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - \gamma \cdot \hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)) = 0 \tag{2}$$

- We need to consider two cases for the term $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$
  - $C1$: If s'_i is non-terminal and $\pi_D(s'_i) = c$ (i.e., $\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)$): Substitute $\phi(s'_i)^T \cdot \boldsymbol{w}^*$ for $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$ in Equation (2)
  - $C2$: If s'_i is a terminal state or $\pi_D(s'_i) = e$ (i.e., $g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}$): Substitute $g(s'_i)$ for $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$ in Equation (2)

LSPI customized for American Options Pricing

- This reduces LSPI Semi-Gradient Equation (1) to:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - \gamma \cdot \hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)) = 0 \tag{2}$$

- We need to consider two cases for the term $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$
  - $C1$: If s'_i is non-terminal and $\pi_D(s'_i) = c$ (i.e., $\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)$): Substitute $\phi(s'_i)^T \cdot \boldsymbol{w}^*$ for $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$ in Equation (2)
  - $C2$: If s'_i is a terminal state or $\pi_D(s'_i) = e$ (i.e., $g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}$): Substitute $g(s'_i)$ for $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$ in Equation (2)
- So rewrite Equation (2) using indicator notation for cases $C1, C2$ as:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - \mathbb{I}_{C1} \cdot \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^* - \mathbb{I}_{C2} \cdot \gamma \cdot g(s'_i)) = 0$$

LSPI customized for American Options Pricing

- This reduces LSPI Semi-Gradient Equation (1) to:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - \gamma \cdot \hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)) = 0 \tag{2}$$

- We need to consider two cases for the term $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$
  - $C1$: If s'_i is non-terminal and $\pi_D(s'_i) = c$ (i.e., $\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)$): Substitute $\phi(s'_i)^T \cdot \boldsymbol{w}^*$ for $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$ in Equation (2)
  - $C2$: If s'_i is a terminal state or $\pi_D(s'_i) = e$ (i.e., $g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}$): Substitute $g(s'_i)$ for $\hat{Q}(s'_i, \pi_D(s'_i); \boldsymbol{w}^*)$ in Equation (2)
- So rewrite Equation (2) using indicator notation for cases $C1, C2$ as:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \boldsymbol{w}^* - \mathbb{I}_{C1} \cdot \gamma \cdot \phi(s'_i)^T \cdot \boldsymbol{w}^* - \mathbb{I}_{C2} \cdot \gamma \cdot g(s'_i)) = 0$$

- Factoring out $\boldsymbol{w}^*$, we get:

$$(\sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{C1} \cdot \gamma \cdot \phi(s'_i))^T) \cdot \boldsymbol{w}^* = \gamma \cdot \sum_i \mathbb{I}_{C2} \cdot \phi(s_i) \cdot g(s'_i)$$

LSPI customized for American Options Pricing

LSPI customized for American Options Pricing

- This can be written in the familiar vector-matrix notation: $\boldsymbol{A} \cdot \boldsymbol{w}^* = \boldsymbol{b}$

$$\boldsymbol{A} = \sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)} \cdot \gamma \cdot \phi(s'_i))^T$$

$$\boldsymbol{b} = \gamma \cdot \sum_i \mathbb{I}_{g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}} \cdot \phi(s_i) \cdot g(s'_i)$$

LSPI customized for American Options Pricing

- This can be written in the familiar vector-matrix notation: $\boldsymbol{A} \cdot \boldsymbol{w}^* = \boldsymbol{b}$

$$\boldsymbol{A} = \sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)} \cdot \gamma \cdot \phi(s'_i))^T$$

$$\boldsymbol{b} = \gamma \cdot \sum_i \mathbb{I}_{g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}} \cdot \phi(s_i) \cdot g(s'_i)$$

- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience (s_i, s'_i) as:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)} \cdot \gamma \cdot \phi(s'_i))^T$$

LSPI customized for American Options Pricing

- This can be written in the familiar vector-matrix notation: $\boldsymbol{A} \cdot \boldsymbol{w}^* = \boldsymbol{b}$

$$\boldsymbol{A} = \sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)} \cdot \gamma \cdot \phi(s'_i))^T$$

$$\boldsymbol{b} = \gamma \cdot \sum_i \mathbb{I}_{g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}} \cdot \phi(s_i) \cdot g(s'_i)$$

- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience (s_i, s'_i) as:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)} \cdot \gamma \cdot \phi(s'_i))^T$$

- m-Vector $\boldsymbol{b}$ is accumulated at each atomic experience (s_i, s'_i) as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \gamma \cdot \mathbb{I}_{g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}} \cdot \phi(s_i) \cdot g(s'_i)$$

LSPI customized for American Options Pricing

- This can be written in the familiar vector-matrix notation: $\boldsymbol{A} \cdot \boldsymbol{w}^* = \boldsymbol{b}$

$$\boldsymbol{A} = \sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)} \cdot \gamma \cdot \phi(s'_i))^T$$

$$\boldsymbol{b} = \gamma \cdot \sum_i \mathbb{I}_{g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}} \cdot \phi(s_i) \cdot g(s'_i)$$

- $m \times m$ Matrix $\boldsymbol{A}$ is accumulated at each atomic experience (s_i, s'_i) as:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{\phi(s'_i)^T \cdot \boldsymbol{w} \geq g(s'_i)} \cdot \gamma \cdot \phi(s'_i))^T$$

- m-Vector $\boldsymbol{b}$ is accumulated at each atomic experience (s_i, s'_i) as:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \gamma \cdot \mathbb{I}_{g(s'_i) > \phi(s'_i)^T \cdot \boldsymbol{w}} \cdot \phi(s_i) \cdot g(s'_i)$$

- Shermann-Morrison incremental inverse of $\boldsymbol{A}$ can be done in $O(m^2)$

Feature functions

Feature functions

- Li, Szepesvari, Schuurmans recommend Laguerre polynomials (first 3)

Feature functions

- Li, Szepesvari, Schuurmans recommend Laguerre polynomials (first 3)
- Over $S' = S_t/K$ where S_t is underlying price and K is strike

Feature functions

- Li, Szepesvari, Schuurmans recommend Laguerre polynomials (first 3)
- Over $S' = S_t/K$ where S_t is underlying price and K is strike
- $\phi_0(S_t) = 1, \phi_1(S_t) = e^{-\frac{S'}{2}}, \phi_2(S_t) = e^{-\frac{S'}{2}} \cdot (1 - S'), \phi_3(S_t) = e^{-\frac{S'}{2}} \cdot (1 - 2S' + S'^2/2)$

Feature functions

- Li, Szepesvari, Schuurmans recommend Laguerre polynomials (first 3)
- Over $S' = S_t/K$ where S_t is underlying price and K is strike
- $\phi_0(S_t) = 1, \phi_1(S_t) = e^{-\frac{S'}{2}}, \phi_2(S_t) = e^{-\frac{S'}{2}} \cdot (1 - S'), \phi_3(S_t) = e^{-\frac{S'}{2}} \cdot (1 - 2S' + S'^2/2)$
- They used these for Longstaff-Schwartz as well as for LSPI

Feature functions

- Li, Szepesvari, Schuurmans recommend Laguerre polynomials (first 3)
- Over $S' = S_t/K$ where S_t is underlying price and K is strike
- $\phi_0(S_t) = 1, \phi_1(S_t) = e^{-\frac{S'}{2}}, \phi_2(S_t) = e^{-\frac{S'}{2}} \cdot (1 - S'), \phi_3(S_t) = e^{-\frac{S'}{2}} \cdot (1 - 2S' + S'^2/2)$
- They used these for Longstaff-Schwartz as well as for LSPI
- For LSPI, we also need feature functions for time

Feature functions

- Li, Szepesvari, Schuurmans recommend Laguerre polynomials (first 3)
- Over $S' = S_t/K$ where S_t is underlying price and K is strike
- $\phi_0(S_t) = 1, \phi_1(S_t) = e^{-\frac{S'}{2}}, \phi_2(S_t) = e^{-\frac{S'}{2}} \cdot (1 - S'), \phi_3(S_t) = e^{-\frac{S'}{2}} \cdot (1 - 2S' + S'^2/2)$
- They used these for Longstaff-Schwartz as well as for LSPI
- For LSPI, we also need feature functions for time
- They recommend $\phi_0^{(t)}(t) = sin(\frac{\pi(T-t)}{2T}), \phi_1^{(t)}(t) = \log(T - t), \phi_2^{(t)}(t) = (\frac{t}{T})^2$

Deep Q-Learning for American Pricing

Deep Q-Learning for American Pricing

- LSPI is data-efficient/compute-efficient, but linearity is a limitation

Deep Q-Learning for American Pricing

- LSPI is data-efficient/compute-efficient, but linearity is a limitation
- Alternative is (incremental) Q-Learning with neural network approx

Deep Q-Learning for American Pricing

- LSPI is data-efficient/compute-efficient, but linearity is a limitation
- Alternative is (incremental) Q-Learning with neural network approx
- We employ the same set up as LSPI (including Experience Replay)

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} f(s; \boldsymbol{w}) & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

where $f(s; \boldsymbol{w})$ is the deep neural network function approximation

Deep Q-Learning for American Pricing

- LSPI is data-efficient/compute-efficient, but linearity is a limitation
- Alternative is (incremental) Q-Learning with neural network approx
- We employ the same set up as LSPI (including Experience Replay)

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} f(s; \boldsymbol{w}) & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

  where $f(s; \boldsymbol{w})$ is the deep neural network function approximation
- Q-Learning update for each atomic experience (s_i, s'_i)

$$\Delta \boldsymbol{w} = \alpha \cdot (\gamma \cdot \hat{Q}(s'_i, \pi(s'_i); \boldsymbol{w}) - f(s_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} f(s_i; \boldsymbol{w})$$

Deep Q-Learning for American Pricing

- LSPI is data-efficient/compute-efficient, but linearity is a limitation
- Alternative is (incremental) Q-Learning with neural network approx
- We employ the same set up as LSPI (including Experience Replay)

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} f(s; \boldsymbol{w}) & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

  where $f(s; \boldsymbol{w})$ is the deep neural network function approximation
- Q-Learning update for each atomic experience (s_i, s'_i)

$$\Delta \boldsymbol{w} = \alpha \cdot (\gamma \cdot \hat{Q}(s'_i, \pi(s'_i); \boldsymbol{w}) - f(s_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} f(s_i; \boldsymbol{w})$$

- When s'_i is a non-terminal state, the update is:

$$\Delta \boldsymbol{w} = \alpha \cdot (\gamma \cdot \max(g(s'_i), f(s'_i; \boldsymbol{w})) - f(s_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} f(s_i; \boldsymbol{w})$$

Deep Q-Learning for American Pricing

- LSPI is data-efficient/compute-efficient, but linearity is a limitation
- Alternative is (incremental) Q-Learning with neural network approx
- We employ the same set up as LSPI (including Experience Replay)

$$\hat{Q}(s, a; \boldsymbol{w}) = \begin{cases} f(s; \boldsymbol{w}) & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

  where $f(s; \boldsymbol{w})$ is the deep neural network function approximation
- Q-Learning update for each atomic experience (s_i, s'_i)

$$\Delta \boldsymbol{w} = \alpha \cdot (\gamma \cdot \hat{Q}(s'_i, \pi(s'_i); \boldsymbol{w}) - f(s_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} f(s_i; \boldsymbol{w})$$

- When s'_i is a non-terminal state, the update is:

$$\Delta \boldsymbol{w} = \alpha \cdot (\gamma \cdot \max(g(s'_i), f(s'_i; \boldsymbol{w})) - f(s_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} f(s_i; \boldsymbol{w})$$

- When s'_i is a terminal state, the update is:

$$\Delta \boldsymbol{w} = \alpha \cdot (\gamma \cdot g(s'_i) - f(s_i; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} f(s_i; \boldsymbol{w})$$

Key Takeaways from this Chapter

Key Takeaways from this Chapter

- Batch RL makes efficient use of data

Key Takeaways from this Chapter

- Batch RL makes efficient use of data
- DQN uses experience replay and fixed Q-learning targets, avoiding the pitfalls of time-correlation and semi-gradient

Key Takeaways from this Chapter

- Batch RL makes efficient use of data
- DQN uses experience replay and fixed Q-learning targets, avoiding the pitfalls of time-correlation and semi-gradient
- LSTD is a direct (gradient-free) solution of Batch TD Prediction

Key Takeaways from this Chapter

- Batch RL makes efficient use of data
- DQN uses experience replay and fixed Q-learning targets, avoiding the pitfalls of time-correlation and semi-gradient
- LSTD is a direct (gradient-free) solution of Batch TD Prediction
- LSPI is an off-policy, experience-replay Control Algorithm using LSTDQ for Policy Evaluation

Key Takeaways from this Chapter

- Batch RL makes efficient use of data
- DQN uses experience replay and fixed Q-learning targets, avoiding the pitfalls of time-correlation and semi-gradient
- LSTD is a direct (gradient-free) solution of Batch TD Prediction
- LSPI is an off-policy, experience-replay Control Algorithm using LSTDQ for Policy Evaluation
- Optimal Exercise of American Options can be tackled with LSPI and Deep Q-Learning algorithms