# 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
- 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

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

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

- 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*

- 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($\lambda$) Prediction

- In Batch TD($\lambda$) 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

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 $\epsilon$-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

- 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:

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

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

# Least-Squares Monte-Carlo (LSMC)

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

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

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

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

- $\boldsymbol{w}^*$ 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:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{\phi}(S_i) \otimes \boldsymbol{\phi}(S_i) \text{ (note: } \otimes \text{ means Outer-Product)}$$

- $m$-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)

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

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\phi(s_i) \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s'_i) \cdot \boldsymbol{w}))^2$$

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

$$\sum_{i=1}^{n} \phi(s_i) \cdot (\phi(s_i) \cdot \boldsymbol{w}^* - (r_i + \gamma \cdot \phi(s'_i) \cdot \boldsymbol{w}^*)) = 0$$

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

  $\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i) \otimes (\phi(s_i) - \gamma \cdot \phi(s'_i))$ (note: $\otimes$ means 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($\lambda$)

- Likewise, we can do LSTD($\lambda$) 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 $\lambda\gamma$

$$\sum_{i=1}^{n} \boldsymbol{E_i} \cdot (\phi(s_i) \cdot \boldsymbol{w^*} - (r_i + \gamma \cdot \phi(s'_i) \cdot \boldsymbol{w}^*)) = 0$$

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

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{E_i} \otimes (\phi(s_i) - \gamma \cdot \phi(s'_i)) \text{ (note: } \otimes \text{ means Outer-Product)}$$

- $m$-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

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

# 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 $\pi$):

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

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

- In each iteration of GPI, we operate with a function approximation

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

- Deterministic policy $\pi_D$ (*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:

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}') &= \sum_i (Q(s_i, a_i; \boldsymbol{w}') - (r_i + \gamma \cdot Q(s'_i, \pi_D(s'_i); \boldsymbol{w}')))^2 \\ &= \sum_i (\phi(s_i, a_i) \cdot \boldsymbol{w}' - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i)) \cdot \boldsymbol{w}'))^2\end{aligned}$$

- $\boldsymbol{w}'$ would give the parameters of the next iteration's function approx

# Least Squares Policy Iteration (LSPI)

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

$$\sum_i \phi(s_i, a_i) \cdot (\phi(s_i, a_i) \cdot \boldsymbol{w}' - (r_i + \gamma \cdot \phi(s'_i, \pi(s'_i)) \cdot \boldsymbol{w}')) = 0$$

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

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \phi(s_i, a_i) \otimes (\phi(s_i, a_i) - \gamma \cdot \phi(s'_i, \pi_D(s'_i)))$$

- $m$-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

| 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

- 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

# Ingredients of Longstaff-Schwartz Algorithm

- $m$ Monte-Carlo paths indexed $i = 0, 1, \ldots, m-1$
- $n+1$ time steps indexed $j = n, n-1, \ldots, 1, 0$ (we move back in time)
- Infinitesimal Risk-free rate at time $t_j$ denoted $r_{t_j}$
- Simulation paths (based on risk-neutral process) of underlying security prices as input 2-dim array $SP[i,j]$
- At each time step, $CF[i]$ is PV of current+future cashflow for path $i$
- $s_{i,j}$ denotes state for $(i,j) :=$ (time $t_j$, price history $SP[i, :(j+1)]$)
- $Payoff(s_{i,j})$ denotes Option Payoff at $(i,j)$
- $\phi_0(s_{i,j}), \ldots, \phi_{r-1}(s_{i,j})$ represent feature functions (of state $s_{i,j}$)
- $w_0, \ldots, w_{r-1}$ are the regression weights
- Regression function $f(s_{i,j}) = w^T \cdot \phi(s_{i,j}) = \sum_{l=0}^{r-1} w_l \cdot \phi_l(s_{i,j})$
- $f(\cdot)$ is estimate of continuation value for in-the-money states

# The Longstaff-Schwartz Algorithm

**Algorithm 0.1:** $\text{LONGSTAFFSCHWARTZ}(SP[0:m, 0:n+1])$

**comment:** $s_{i,j}$ is shorthand for state at $(i,j) := (t_j,\ SP[i,:(j+1)])$

$CF[0:m] \leftarrow [Payoff(s_{i,n})$ for $i$ in range$(m)]$

**for** $j \leftarrow n-1$ **downto** $1$

**do**
$$\begin{cases} CF[0:m] \leftarrow CF[0:m] \cdot e^{-r_{t_j}(t_{j+1}-t_j)} \\ X \leftarrow [\phi(s_{i,j}) \text{ for } i \text{ in range}(m) \text{ if } Payoff(s_{i,j}) > 0] \\ Y \leftarrow [CF[i] \text{ for } i \text{ in range}(m) \text{ if } Payoff(s_{i,j}) > 0] \\ w \leftarrow (X^T \cdot X)^{-1} \cdot X^T \cdot Y \\ \textbf{comment: } \text{Above regression gives estimate of continuation value} \\ \textbf{for } i \leftarrow 0 \textbf{ to } m-1 \\ \quad \textbf{do } CF[i] \leftarrow Payoff(s_{i,j}) \textbf{ if } Payoff(s_{i,j}) > w^T \cdot \phi(s_{i,j}) \end{cases}$$

$exercise \leftarrow Payoff(s_{0,0})$

$continue \leftarrow e^{-r_0(t_1-t_0)} \cdot mean(CF[0:m])$

**return** $(\max(exercise, continue))$

# RL 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 outline two RL Algorithms:
  - Least Squares Policy Iteration (LSPI)
  - Fitted Q-Iteration (FQI)
- Both Algorithms are batch methods solving a linear system
- Reference: Li, Szepesvari, Schuurmans paper

# LSPI customized for American Options Pricing

- $a$ is $e$ (exercise) or $c$ (continue), $s$ is $s_{i,j}$, $s'$ is $s_{i,j+1}$
- $r$ is $\gamma \cdot Payoff(s_{i,j+1})$ if $\pi(s_{i,j+1}) = e$ and $r = 0$ if $\pi(s_{i,j+1}) = c$
- We set $Q(s_{i,j}, e) = Payoff(s_{i,j})$ (not to be learnt)
- We set $Q(s_{i,j}, c; w) = w^T \cdot \phi(s_{i,j})$ (to be learnt)
- This requires us to set: $x(s_{i,j}, c) = \phi(s_{i,j})$ and $x(s_{i,j}, e) = 0$
- When $\pi(s_{i,j+1}) = c$, i.e., when $w^T \cdot \phi(s_{i,j+1}) \geq Payoff(s_{i,j+1})$
    - $A$ update is: $\phi(s_{i,j}) \cdot (\phi(s_{i,j}) - \gamma \cdot \phi(s_{i,j+1}))^T$
    - $B$ update is: $0$
- When $\pi(s_{i,j+1}) = e$, i.e., when $w^T \cdot \phi(s_{i,j+1}) < Payoff(s_{i,j+1})$
    - $A$ update is: $\phi(s_{i,j}) \cdot (\phi(s_{i,j}) - \gamma \cdot 0)^T$
    - $B$ update is: $\gamma \cdot Payoff(s_{i,j+1}) \cdot \phi(s_{i,j})$

**Algorithm 0.2:** $\text{LSPI-AMERICANPRICING}(SP[0:m, 0:n+1])$

**comment:** $s_{i,j}$ is shorthand for state at $(i,j) := (t_j, SP[i, :(j+1)])$

**comment:** $A$ is an $r \times r$ matrix, $b$ and $w$ are $r$-length vectors

**comment:** $A_{i,j} \leftarrow \phi(s_{i,j}) \cdot (\phi(s_{i,j}) - \gamma \cdot \mathbb{I}_{w^T \cdot \phi(s_{i,j+1}) \geq Payoff(s_{i,j+1})} \cdot \phi(s_{i,j+1}))^T$

**comment:** $b_{i,j} \leftarrow \gamma \cdot \mathbb{I}_{w^T \cdot \phi(s_{i,j+1}) < Payoff(s_{i,j+1})} \cdot Payoff(s_{i,j+1}) \cdot \phi(s_{i,j})$

$A \leftarrow 0, B \leftarrow 0, w \leftarrow 0$

**for** $i \leftarrow 0$ **to** $m - 1$

$$
\textbf{do} \begin{cases}
\textbf{for } j \leftarrow 0 \textbf{ to } n-1 \\
\quad \textbf{do} \begin{cases}
Q \leftarrow Payoff(s_{i,j+1}) \\
P \leftarrow \phi(s_{i,j+1}) \textbf{ if } j < n-1 \textbf{ and } Q \leq w^T \cdot \phi(s_{i,j+1}) \textbf{ else } 0 \\
R \leftarrow Q \textbf{ if } Q > w^T \cdot P \textbf{ else } 0 \\
A \leftarrow A + \phi(s_{i,j}) \cdot (\phi(s_{i,j}) - e^{-r_{t_j}(t_{j+1}-t_j)} \cdot P)^T \\
B \leftarrow B + e^{-r_{t_j}(t_{j+1}-t_j)} \cdot R \cdot \phi(s_{i,j})
\end{cases} \\
w \leftarrow A^{-1} \cdot b, A \leftarrow 0, b \leftarrow 0 \textbf{ if } (i+1)\% BatchSize == 0
\end{cases}
$$

# Fitted Q-Iteration for American Options Pricing

**Algorithm 0.3:** FQI-AmericanPricing($SP[0:m, 0:n+1]$)

**comment:** $s_{i,j}$ is shorthand for state at $(i,j) := (t_j, SP[i, :(j+1)])$

**comment:** $A$ is an $r \times r$ matrix, $b$ and $w$ are $r$-length vectors

**comment:** $A_{i,j} \leftarrow \phi(s_{i,j}) \cdot \phi(s_{i,j})^T$

**comment:** $b_{i,j} \leftarrow \gamma \cdot \max(Payoff(s_{i,j+1}), w^T \cdot \phi(s_{i,j+1})) \cdot \phi(s_{i,j})$

$A \leftarrow 0, B \leftarrow 0, w \leftarrow 0$

**for** $i \leftarrow 0$ **to** $m-1$

**do** $\begin{cases} \textbf{for } j \leftarrow 0 \textbf{ to } n-1 \\ \quad \textbf{do} \begin{cases} Q \leftarrow Payoff(s_{i,j+1}) \\ P \leftarrow \phi(s_{i,j+1}) \textbf{ if } j < n-1 \textbf{ else } 0 \\ A \leftarrow A + \phi(s_{i,j}) \cdot \phi(s_{i,j})^T \\ B \leftarrow B + e^{-r_{t_j}(t_{j+1}-t_j)} \cdot \max(Payoff(s_{i,j+1}), w^T \cdot P) \cdot \phi(s_{i,j}) \end{cases} \\ w \leftarrow A^{-1} \cdot b, A \leftarrow 0, b \leftarrow 0 \textbf{ if } (i+1)\% BatchSize == 0 \end{cases}$

# 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 and FQI
- For LSPI and FQI, we also need feature functions for time
- They recommend $\phi_0^t(t) = sin(\frac{\pi(T-t)}{2T}), \phi_1^t(t) = \log(T - t), \phi_2^t(t) = (\frac{t}{T})^2$

# 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