- CME 241 Notes:
  - Incremental RL Makes inefficient use of training data
    - ↳ Incremental MC estimates $V(s;w)$ using $\nabla_w L(w)$ ⟵ Gradient descent
    - ↳ You would call update method in function approx.
      - ↳ Inefficient use of each data point. Point is discarded after each update
  - Batch Monte Carlo Prediction
    - ↳ Given batch data (s_i, G_i)
    - ↳ Use solve method of FunctionApprox
      - ↳ Given data, find best fitting weights
    - ↳ You can store batch of data in database and randomly sample points with replacement
      - ↳ Use the same datapoints again and again.
      - ↳ Converges to weight w^*
    - ↳ Reuse of atomic or trace experience is experience replay
      - ↳ Makes efficient use of training data D

$$D = [(s_i, R_i, s_i') \mid 1 \leq i \leq n]$$

        - ↳ In code, these are "Transition Step"
    - ↳ Experience Replay on finite experiences D
  - Batch TD(λ) Prediction

$$D = [(s_{i,0}, R_{i,1}, s_{i,1}, R_{i,2}, \ldots) \mid 1 \leq i \leq n]$$

    - ↳ Randomly pick trace experiences from D

- **The Deep Q-Networks (DQN) Control Algorithm**
  - ↳ Uses Experience Replay and fixed Q-learning targets ⟵ Two innovations
  - ↳ One of the first works to use experience replay
  - ↳ At each t from each episode:
    1. Given state S_t, take action A_t according to ϵ-greedy policy extracted from Q-network values $Q(S_t, a; w)$
    2. Obtain reward R_{t+1} and next state S_{t+1}
    3. Store atomic experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory D
    4. Sample random minibatch of atomic experiences
    5. Update Q-network parameters w using Q-learning targets based on "frozen" parameters w^- of target network
       ↑ This is updated every x steps.
  - ↳ Clever techniques / variants not explicitly proven : "hacky"

- **Least Squares RL Prediction**
  - ↳ Batch RL Prediction for general function approx. is iterative
    - ↳ Uses experience replay and gradient descent.
  - ↳ We can use a direct solve (without gradients) for linear function approx
    - ↳ Least Squares Mean Costs

$$L(w) = \frac{1}{2n} \sum_{i=0}^{n} \left(\phi(s_i)^T \cdot w - G_i\right)^2$$

      - ↳ Set gradient of L wrt w : 0 and solve for w^*

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

      - ↳ See slide 9 for development of A, b
    - ↳ Least Squares Temporal Difference : Slide 1

A is formed from outer product of features (see A update rule slide 7)

↳ Updated incrementally for efficient Sherman - Morrison Inverse

↳ Least Squares Algorithm limited to linear function approx

↳ Recommended if you can get low features and linear function approx.

↳ In reality, this is impractical.

↳ Q-learning is off policy (experience pairs coming from database of past experience).

↳ Approach also used linear function approx

↳ Semigradient as in TD avoids bootstrapping, which would complete the deadly triad.

↳ LSTD(λ) is a batch form of TD(λ) using eligibility traces

Accumulator $\nabla_w V(s; w) = \phi(s)$

E_i = Eligibility trace at atomic experience i:

$$A \longleftarrow A + E_i \cdot \left(\phi(s) - \gamma \cdot \phi(s')\right)^T$$

Tracks how many times feature occurs in the past X timesteps weighted by recency.

Implement LSTD(λ) algorithm, building A, b incrementally

- **Least Squares RL Control**
  - Policy Evaluation as Least-Squares Q-value Prediction
  - Greedy Policy Improvement
  - Least-Squares Policy Iteration (slides 13-13)
    1. Get $Q(s, a; w)$ using function approximation with initial features, weights

② Get Greedy policy to maximize Q

③ Sample minibatch of experiences

④ Goal of iteration to solve weights w^* to minimize $\mathcal{L}(w)$

↳ Using Semigradient

↳ See slide 13

- We learned about LSTD batch for Q-value function (LSTDQ)
  - ↳ This sits inside LSPI for control problem.
  - ↳ LSPI is a linear function approximation
    - ↳ Fluctuates about optimum in linear function approximation

- <u>LSPI for Optimal Exercise of American Options</u>
  - LSPI can be predictive of Complex American Option
  - Recall, it is never optimum to exercise early for an American Call Option
  - LSPI converges much faster than Longstaff-Schwartz algorithm.
  - Only use linear function approx when you are holding the option.
    - ↳ When exercising, you know the true value
  - Each data point in atomic experiences database as: $(s, c, 0, s')$

    (c ← Only store continue (hold) action option)

    (0 ← You will not get a reward when you hold.)

  - Citibank project on exotic options