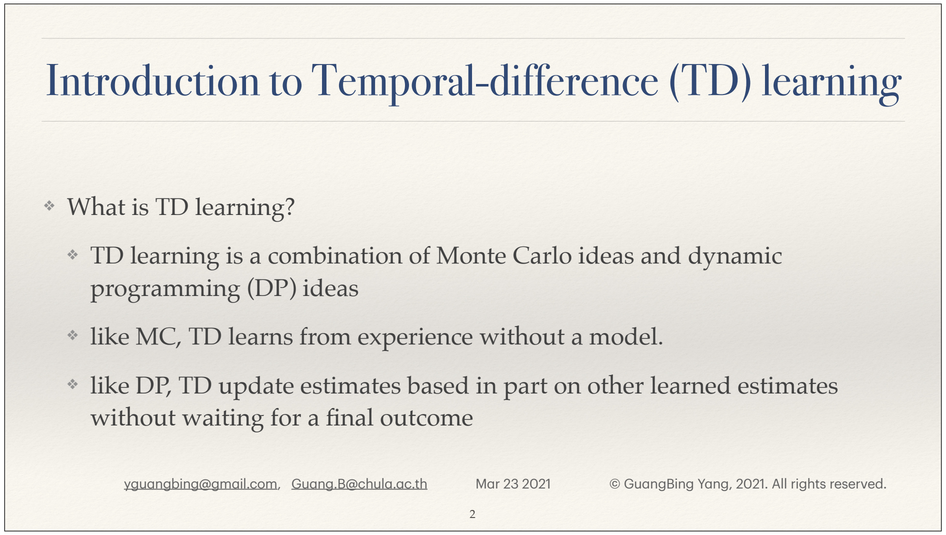
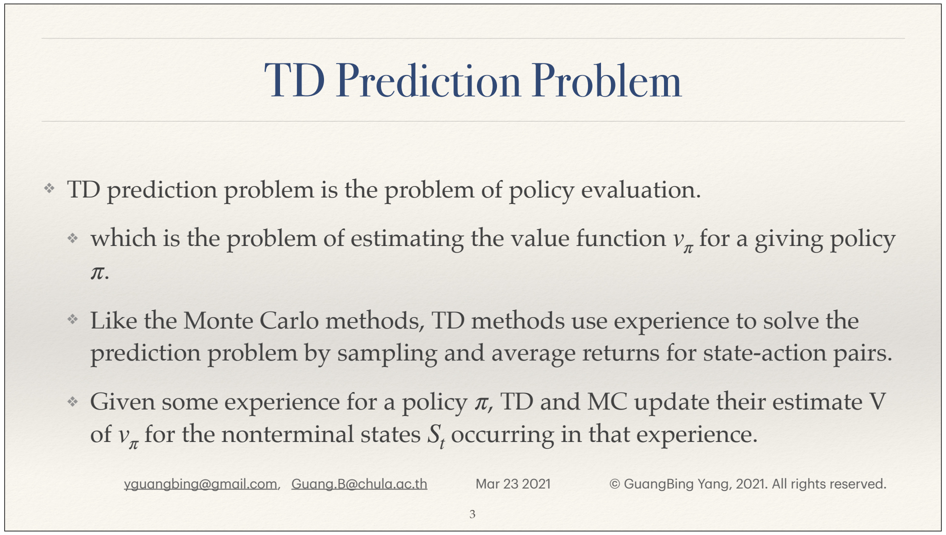
Lecture 9 - Temporal-difference (TD) learning

Instructor: GuangBing Yang, PhD



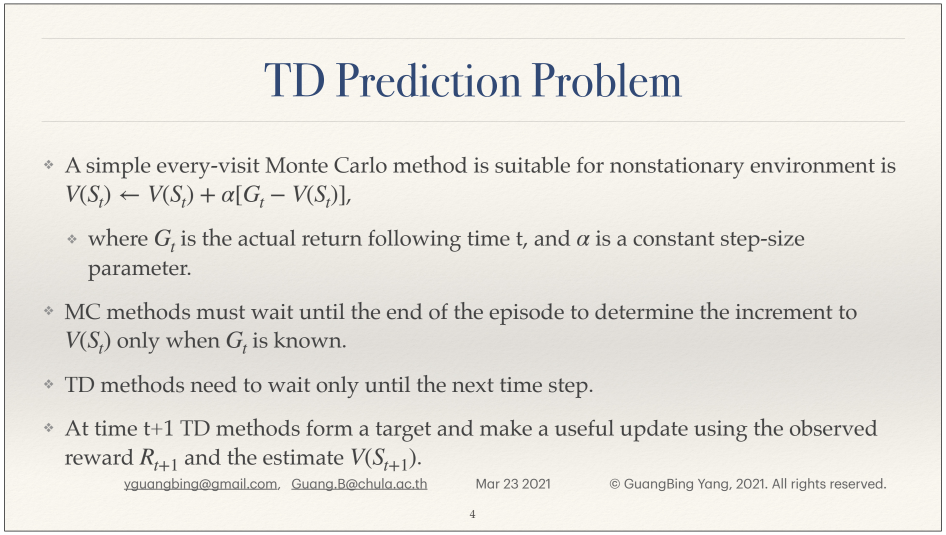
Introduction to Temporal-difference (TD) learning

- What is TD learning?
  - TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas
  - like MC, TD learns from experience without a model.
  - like DP, TD update estimates based in part on other learned estimates without waiting for a final outcome



TD Prediction Problem

- TD prediction problem is the problem of policy evaluation.
  - which is the problem of estimating the value function v_π for a giving policy π.
  - Like the Monte Carlo methods, TD methods use experience to solve the prediction problem by sampling and average returns for state-action pairs.
  - Given some experience for a policy π, TD and MC update their estimate V of v_π for the nonterminal states S_t occurring in that experience.



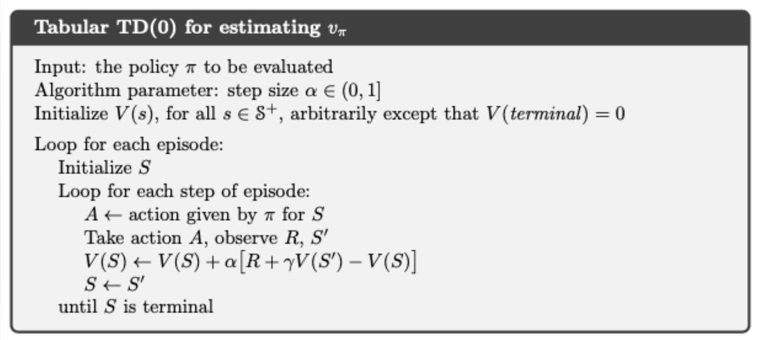
TD Prediction Problem

- A simple every-visit Monte Carlo method is suitable for nonstationary environment is $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$,
  - where G_t is the actual return following time t, and α is a constant step-size parameter.
- MC methods must wait until the end of the episode to determine the increment to $V(S_t)$ only when G_t is known.
- TD methods need to wait only until the next time step.
- At time t+1 TD methods form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$.

TD Prediction Problem

- Which is $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$, immediately on transition to S_{t+1} and receiving R_{t+1}.
- The target for the Monte Carlo update is G_t,
- The target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$.
- Because it is immediately on next step, so it is called TD(0), or one-step TD,
- It is a special case of the $TD(\lambda)$ and n-step TD methods.

Tabular TD(0) algorithm for estimating v_π



Tabular TD(0) for estimating v_π

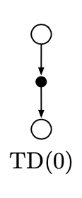
```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$
Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    $A \leftarrow$ action given by $\pi$ for $S$
    Take action $A$, observe $R$, $S'$
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until $S$ is terminal
```

Tabular TD(0) algorithm for estimating v_π

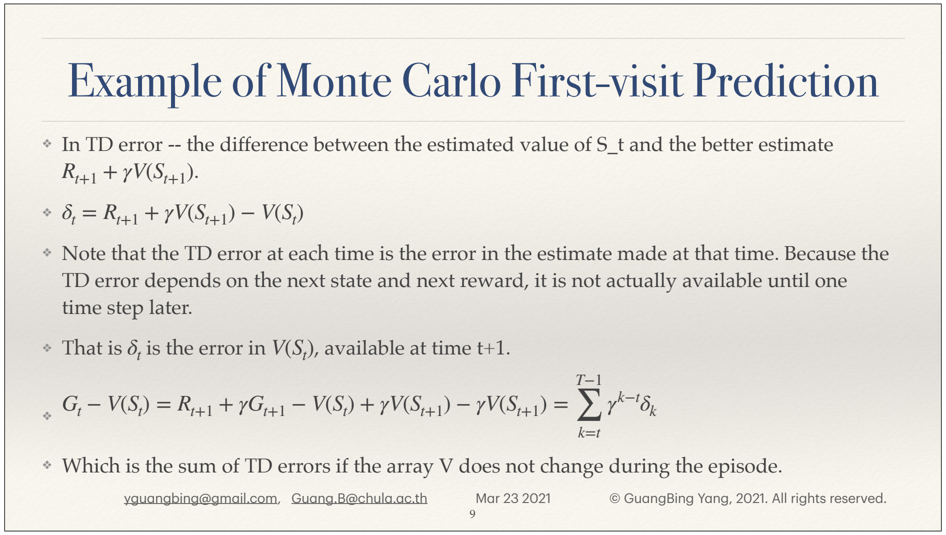
- Update of TD(0) is based on part of existing estimate, so it is a bootstrapping method (like DP)
- The Monte Carlo updates expected value after the termination because the expected value is unknown until the end of the episode, $v_\pi = E_\pi[G_t \mid S_t = s]$.
- The DP target is an estimate from a model of the environment not from the expected values. Because $v_\pi(S_{t+1})$ is not known, use current estimate $V(S_{t+1})$ instead.
- For DP, $v_\pi = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$
- TD target is an estimate for both: samples the expected and uses the current estimate V. So, $v_\pi = V(S_t) + E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$

Tabular TD(0) algorithm for estimating v_π

- Backup diagram of TD(0) is shown on the right:
  - The value estimate for the state node at the top of the backup diagram is updated on the basis of the one sample transition from it to the immediately following state.
  - 0Sample updates differ from the expected updates of DP methods in that they are based on a single sample successor rather than on a complete distribution of all possible successors.

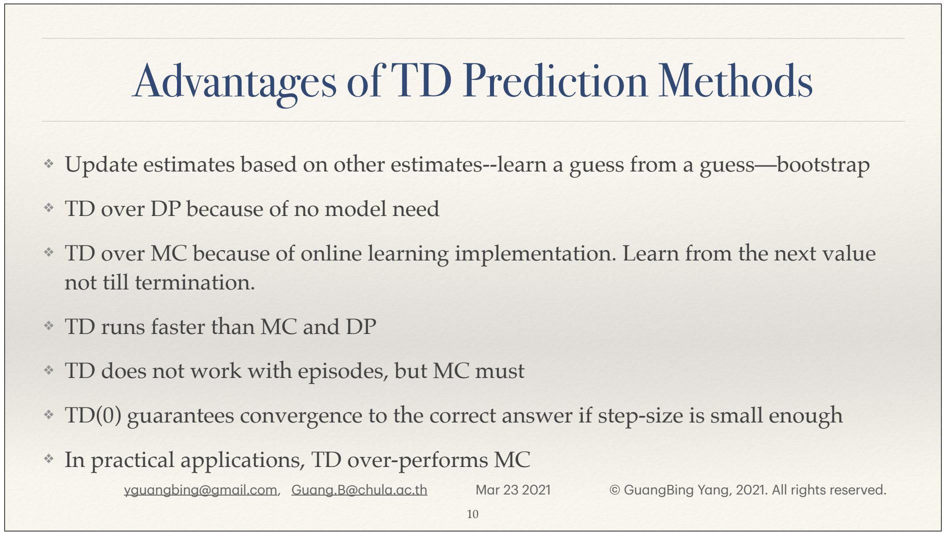


TD(0) backup diag.



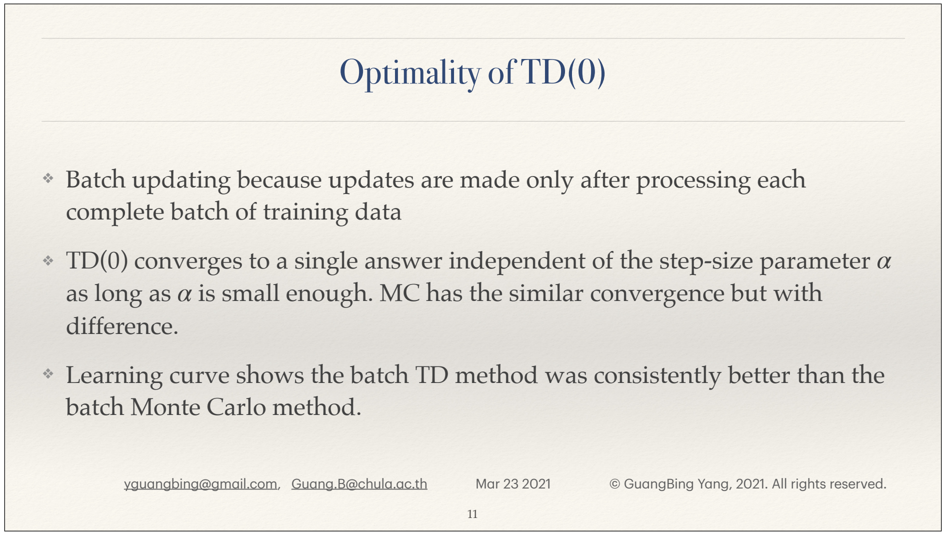
Example of Monte Carlo First-visit Prediction

- In TD error -- the difference between the estimated value of S_t and the better estimate $R_{t+1} + \gamma V(S_{t+1})$.
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
- Note that the TD error at each time is the error in the estimate made at that time. Because the TD error depends on the next state and next reward, it is not actually available until one time step later.
- That is δ_t is the error in $V(S_t)$, available at time t+1.
- $G_t - V(S_t) = R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) = \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k$
- Which is the sum of TD errors if the array V does not change during the episode.



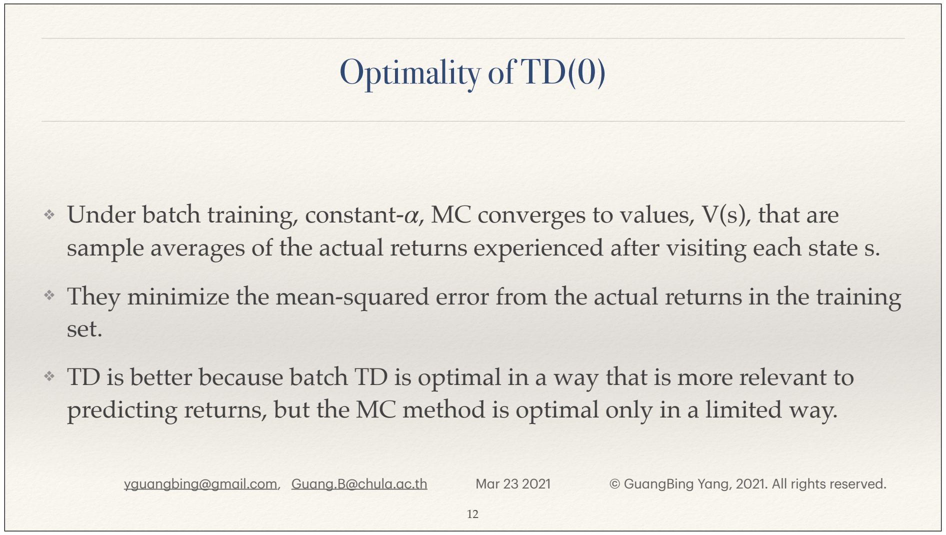
Advantages of TD Prediction Methods

- Update estimates based on other estimates--learn a guess from a guess—bootstrap
- TD over DP because of no model need
- TD over MC because of online learning implementation. Learn from the next value not till termination.
- TD runs faster than MC and DP
- TD does not work with episodes, but MC must
- TD(0) guarantees convergence to the correct answer if step-size is small enough
- In practical applications, TD over-performs MC



Optimality of TD(0)

- Batch updating because updates are made only after processing each complete batch of training data
- TD(0) converges to a single answer independent of the step-size parameter α as long as α is small enough. MC has the similar convergence but with difference.
- Learning curve shows the batch TD method was consistently better than the batch Monte Carlo method.



Optimality of TD(0)

- Under batch training, constant-α, MC converges to values, V(s), that are sample averages of the actual returns experienced after visiting each state s.
- They minimize the mean-squared error from the actual returns in the training set.
- TD is better because batch TD is optimal in a way that is more relevant to predicting returns, but the MC method is optimal only in a limited way.

Optimality of TD(0)

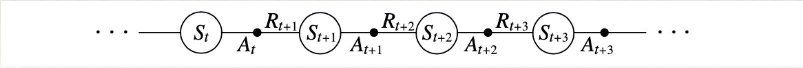
- A general difference between the estimates of batch TD(0) and batch MC:
  - Batch MC estimates minimize mean-squared error on the training set
  - TD(0) estimates for the maximum-likelihood of the markov process
  - In general, the maximum-likelihood estimate of a parameter is the parameter value whose probability of generating the data is greatest.
  - The certainty-equivalence estimate is the reason that TD methods converge quickly than MC

Sarsa: On-policy TD control

- Sarsa stands for “State, action, reward, state, action.”
- Use TD prediction methods for the control problem. Follow the GPI (generalized policy iteration) using TD methods for the evaluation or prediction.
- In MC methods, there are needs for trade-off exploration and exploitation--on and off policy. TD also has this trade-off -- on-policy or off-policy.
- First step, learn an action-value function rather than a state-value function. On-policy method, estimate $q_\pi(s, a)$ for current behaviour policy π in b for all states and actions a.

Sarsa: On-policy TD control

- Use TD prediction methods for the control problem. Follow the GPI (generalized policy iteration) using TD methods for the evaluation or prediction.
- In MC methods, there are needs for trade-off exploration and exploitation--on and off policy. TD also has this trade-off-- on- or off- policy.
- First step, learn an action-value function rather than a state-value function. On-policy method, estimate $q_\pi(s, a)$ for current behaviour policy π in b for all states and actions a.



Sarsa

Sarsa: On-policy TD control

- Previously, considered transitions from state to state and learned the values of states.
- Now, consider transitions from state-action pair to state-action pair, and learn the values of state-action pairs.
- They are both Markov chains with a reward process. The theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm for action values:

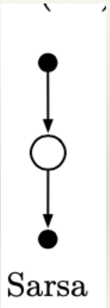
Sarsa: On-policy TD control

- This update is done after every transition from a nonterminal stateS_t.
- If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is defined as zero.
- This rule uses every element of the quintuple of events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state-action pair to the next.
- This quintuple gives rise to the name Sarsa for the algorithm
- The Sarsa algorithm is about:

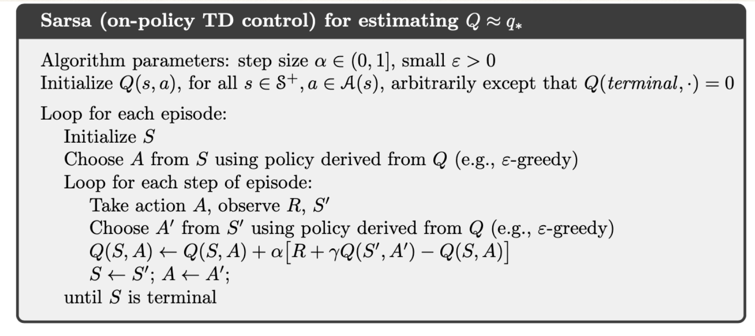
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Sarsa: On-policy TD control

- The backup diagram of Sarsa is given on the right:
- As in all on-policy methods, we continually estimate $q_\pi$for the behaviour policy π, and at the same time change π toward greediness with respect to q_π. The general form of the Sarsa control algorithm is given in the next slide.



Sarsa: On-policy TD control



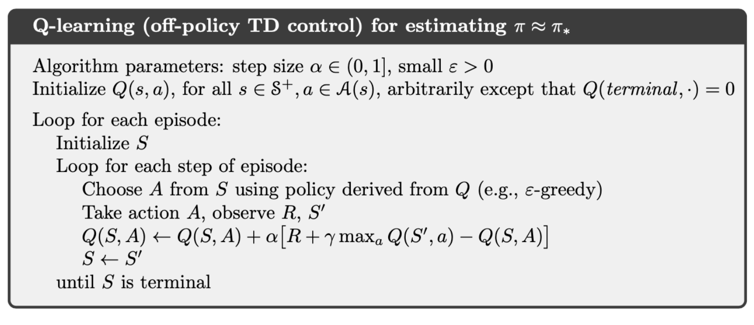
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Loop for each step of episode:
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal
```

Q-learning: Off-policy TD Control

- Off-policy TD control algorithm is also known as Q-learning defined by
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
- The learned action-value function Q directly approximate q_*, the optimal action-value function, independent of the policy being followed.
- The policy determines which state-action pairs are visited and updated.
- All is required for correct convergence is that all pairs continue to be updated.
- Q converges with probability 1 to q_*.

Q-learning: Off-policy TD Control

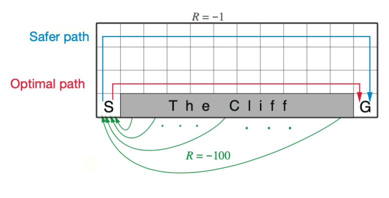


```
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Q-learning Algorithm

Example of Q-learning

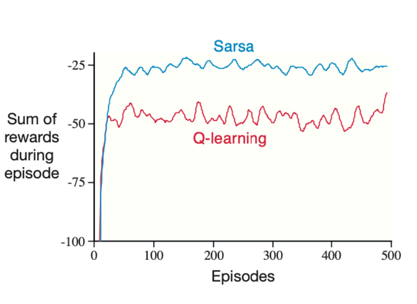
- Cliff Walking, a grid world example compares Sarsa and Q-learning.



Cliff-walking Q-learning Example

Example of Q-learning

- The performance of the Sarsa and Q-learning methods with ϵ-greedy action selection, $\epsilon = 0.1$



Expected Sarsa

- The learning algorithm is just like Q-learning except that instead of the maximum over next state-action pairs it uses the expected value, taking into account how likely each action is under the current policy. That is, consider the algorithm with the update rule.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma E_\pi[Q(S_{t+1}, A_{t+1}) \,|\, S_{t+1}] - Q(S_t, A_t)]$$

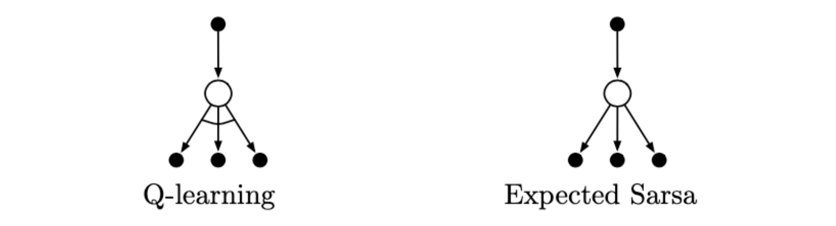
- $$\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a \,|\, S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Expected Sarsa

- Given the next state S_{t+1}, this algorithm moves deterministically in the same direction as Sarsa moves in expectation, so called expected Sarsa.
- Expected Sarsa is more complex computationally than Sarsa but in return, it eliminates the variance due to the random selection of A_{t+1}. It performs slightly better than Sarsa.

Expected Sarsa

- Backup diagrams of Q-learning and Expected Sarsa



Expected Sarsa

- Expected Sarsa can safely set $\alpha = 1$ without suffering any degradation of asymptotic performance
- whereas Sarsa can only perform well in the long run at a small value of α, at which short-term performance is poor.
- There is a consistent empirical advantage of Expected Sarsa over Sarsa.

Maximization Bias and Double Learning

- All the control algorithms maximize their target policies. E.g., in Q-learning, the target policy is the greedy policy given the current action values;
- In Sarsa, the policy is ϵ-greedy.
- The maximum of the true is zero, but the maximum of the estimates is positive, a positive bias. Call this maximization bias.

Maximization Bias and Double Learning

- An Algorithm to avoid maximization bias is double Q-learning.
- Divided the plays in two sets to learn two independent estimates Q1 and Q2.
- Use Q1 to determine the maximizing action $A^* = argmax_a Q_1(a)$
- Q2 provides the estimate of its value $Q_2(A^*) = Q_2(argmax_a Q_1(a))$.
- Repeat above process reversely with $Q_1(A^*) = Q_1(argmax_a Q_2(a))$.

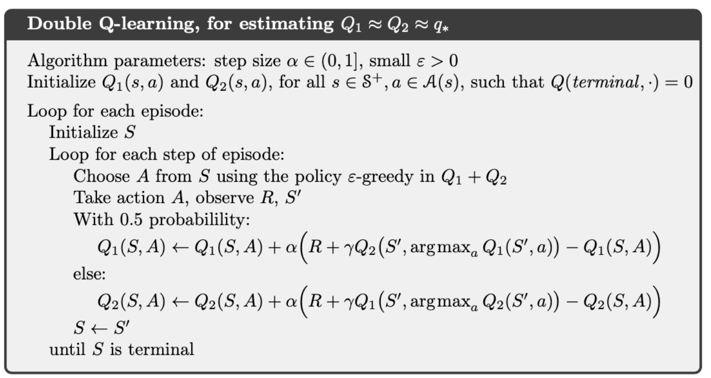
Maximization Bias and Double Learning

- This is the idea of double learning. Note that although there are two estimates, but only one estimate is updated on each play; double learning doubles the memory requirements, but does not increase the amount of computation per step.
- The idea of double learning extends naturally to algorithms for full MDPs. For example, the double learning algorithm analogous to Q-learning, called Double Q-learning, divides the time steps in two, perhaps by flipping a coin on each step. If the coin comes up heads, the update is

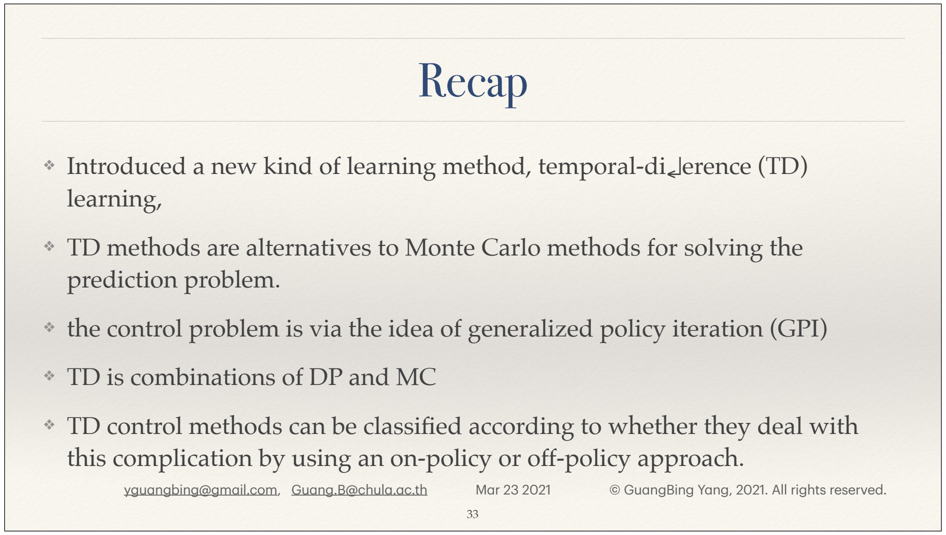
Maximization Bias and Double Learning

- Like flip a coin, if the coin comes up faces, update done for Q1
- $Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \text{argmax}_a Q_1(S_{t+1}, a) - Q_1(S_t, A_t)]$
- if the coin comes up tails, update Q2 as:
- $Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \text{argmax}_a Q_2(S_{t+1}, a) - Q_2(S_t, A_t)]$

Maximization Bias and Double Learning

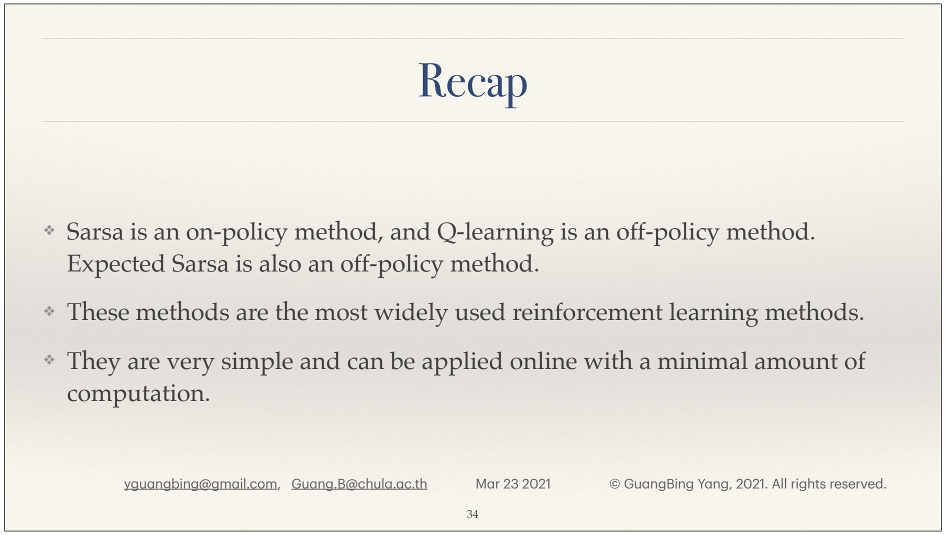


```
Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
    Take action $A$, observe $R$, $S'$
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \text{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \text{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until $S$ is terminal
```



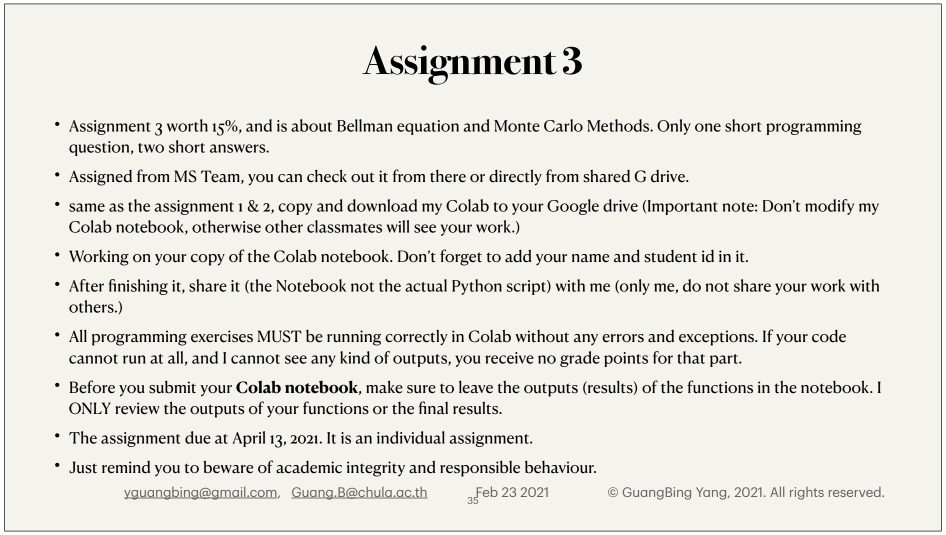
Recap

- Introduced a new kind of learning method, temporal-difference (TD) learning,
- TD methods are alternatives to Monte Carlo methods for solving the prediction problem.
- the control problem is via the idea of generalized policy iteration (GPI)
- TD is combinations of DP and MC
- TD control methods can be classified according to whether they deal with this complication by using an on-policy or off-policy approach.



Recap

- Sarsa is an on-policy method, and Q-learning is an off-policy method. Expected Sarsa is also an off-policy method.
- These methods are the most widely used reinforcement learning methods.
- They are very simple and can be applied online with a minimal amount of computation.



Assignment 3

- Assignment 3 worth 15%, and is about Bellman equation and Monte Carlo Methods. Only one short programming question, two short answers.
- Assigned from MS Team, you can check out it from there or directly from shared G drive.
- same as the assignment 1 & 2, copy and download my Colab to your Google drive (Important note: Don't modify my Colab notebook, otherwise other classmates will see your work.)
- Working on your copy of the Colab notebook. Don't forget to add your name and student id in it.
- After finishing it, share it (the Notebook not the actual Python script) with me (only me, do not share your work with others.)
- All programming exercises MUST be running correctly in Colab without any errors and exceptions. If your code cannot run at all, and I cannot see any kind of outputs, you receive no grade points for that part.
- Before you submit your **Colab notebook**, make sure to leave the outputs (results) of the functions in the notebook. I ONLY review the outputs of your functions or the final results.
- The assignment due at April 13, 2021. It is an individual assignment.
- Just remind you to beware of academic integrity and responsible behaviour.

Questions and Lab