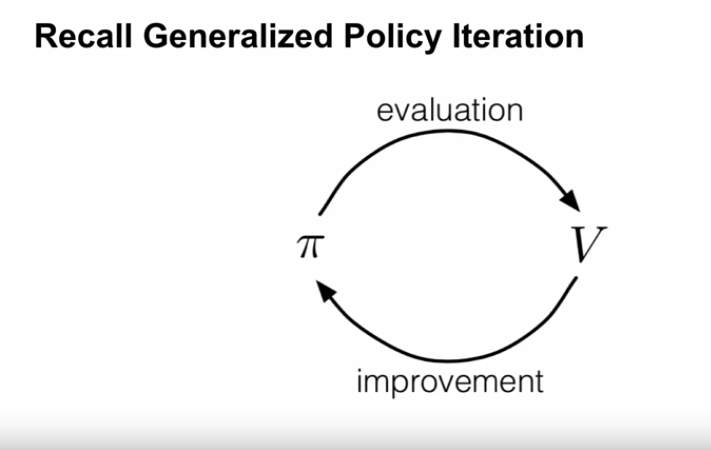
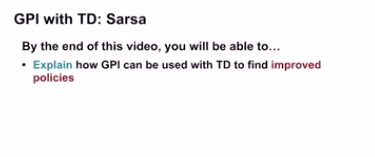
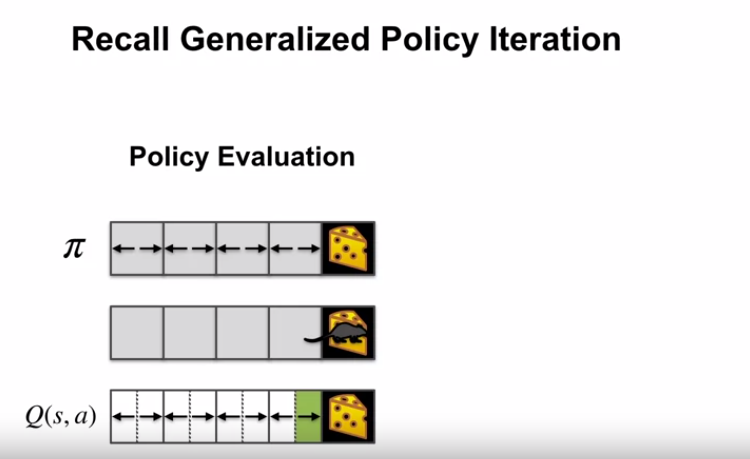
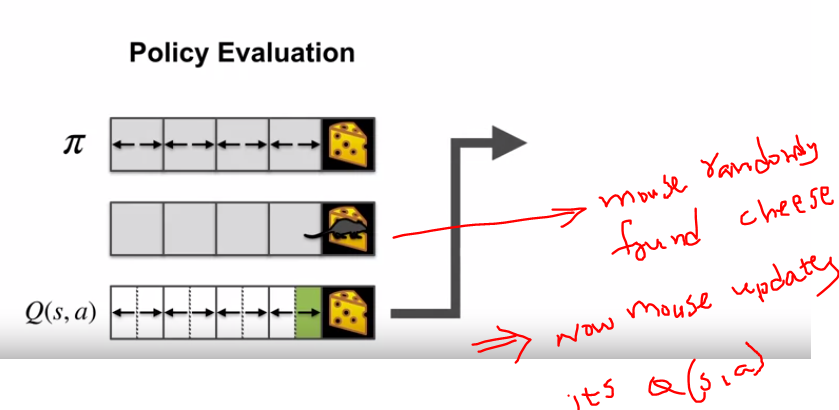
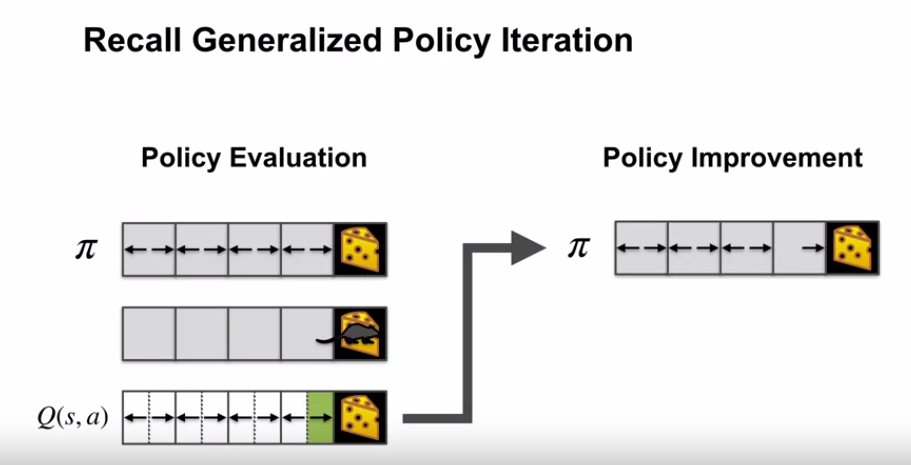
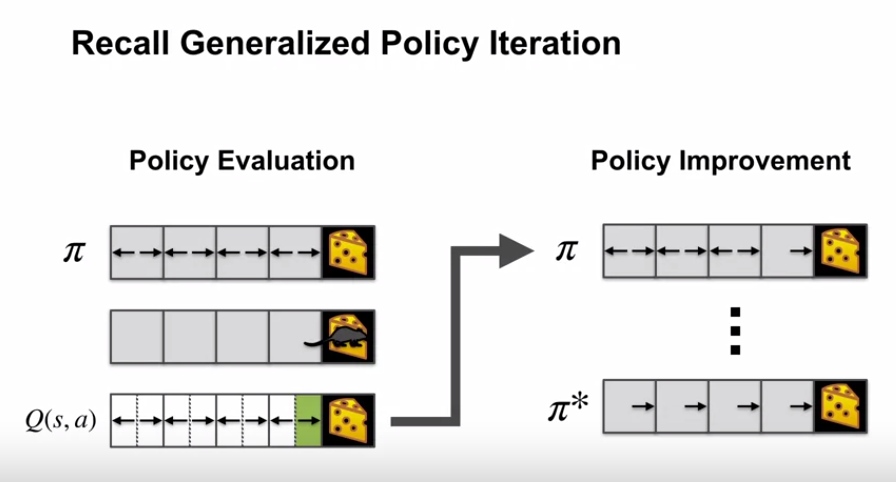
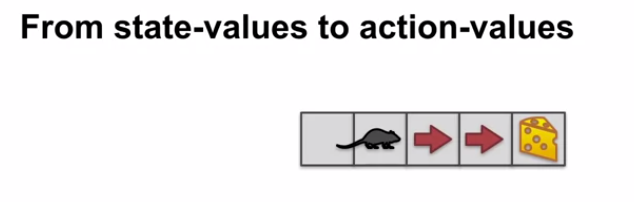
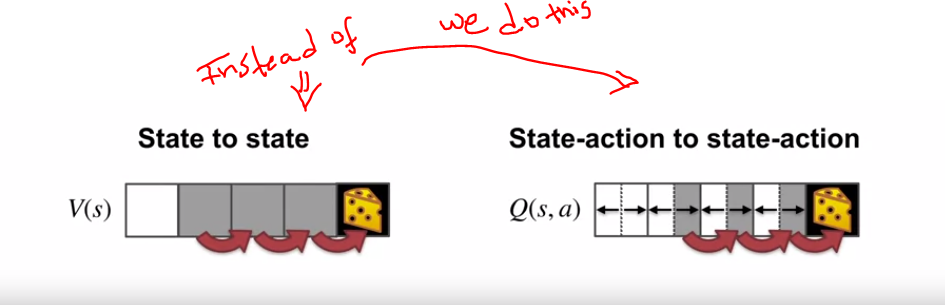
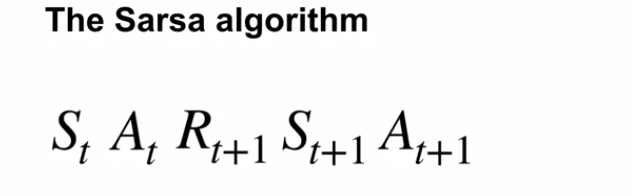
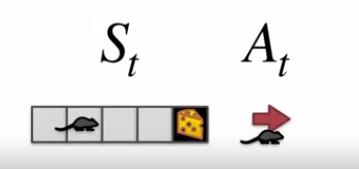
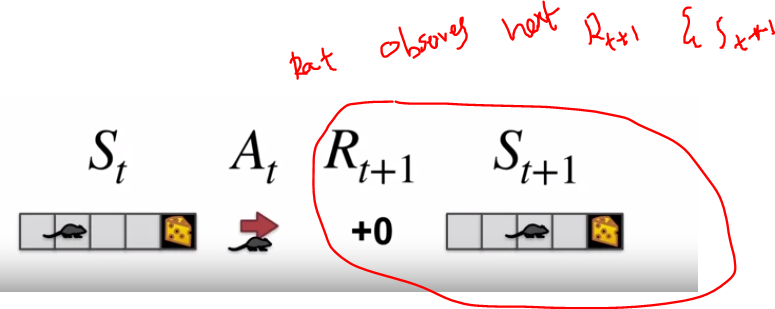
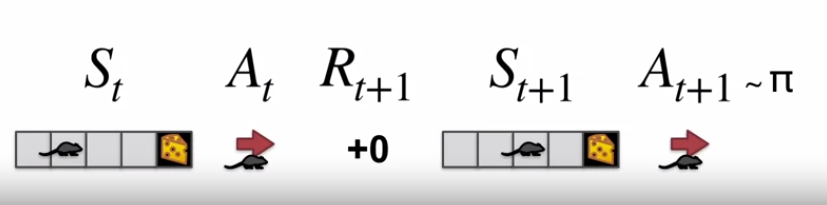
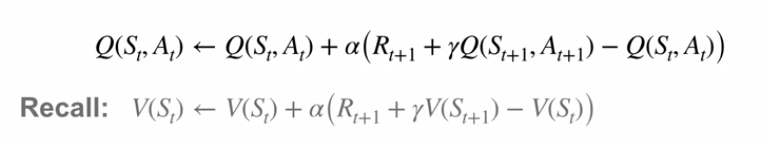
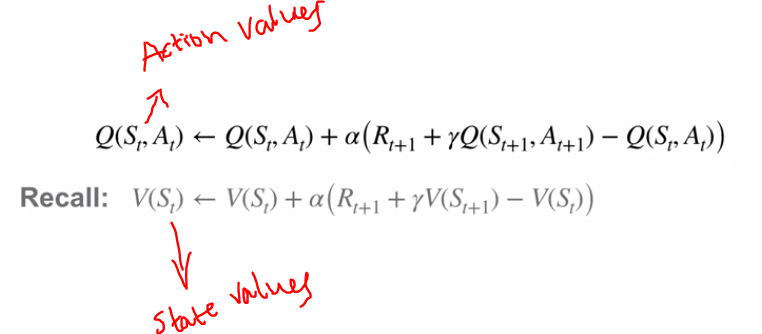
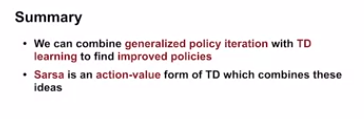
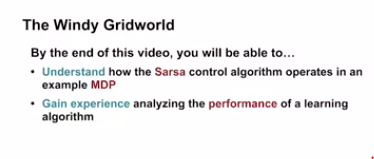
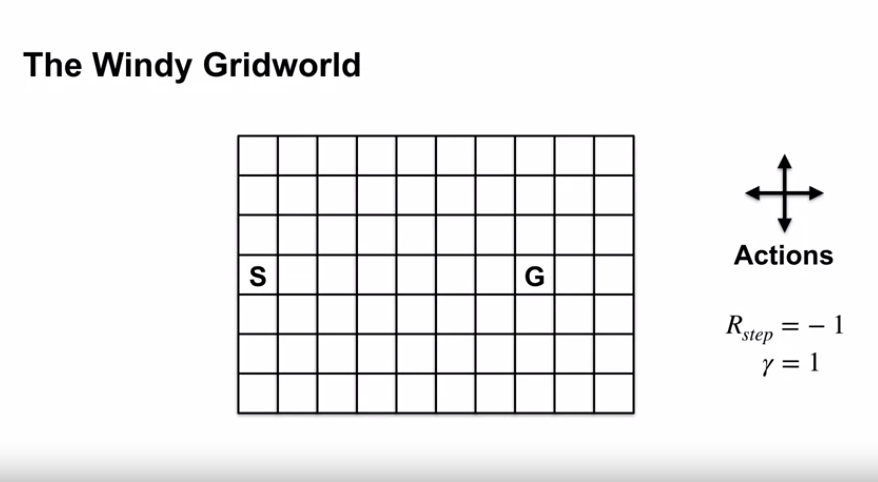
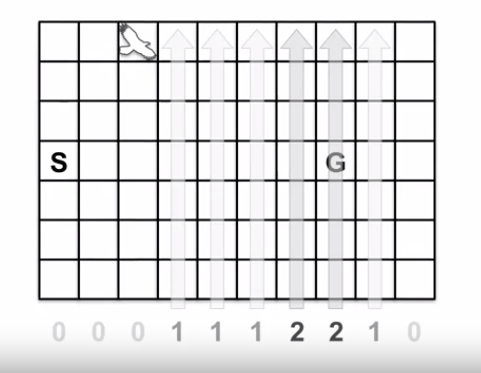
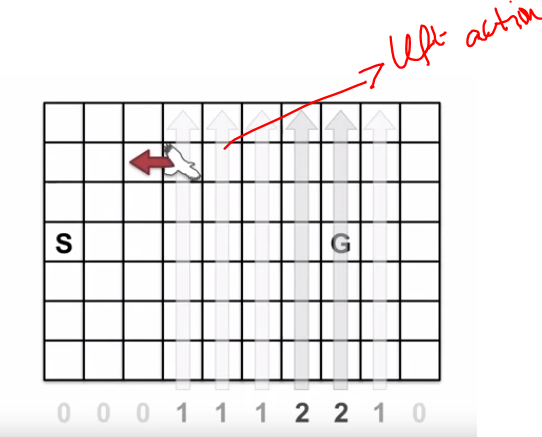
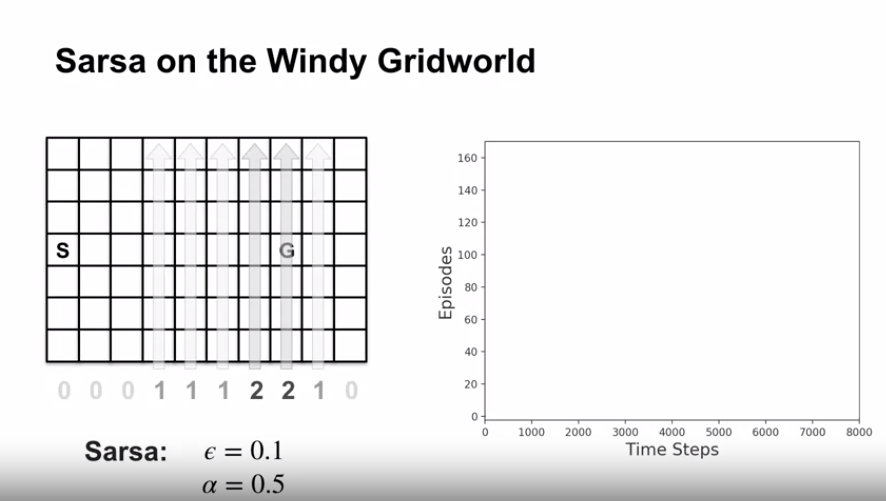
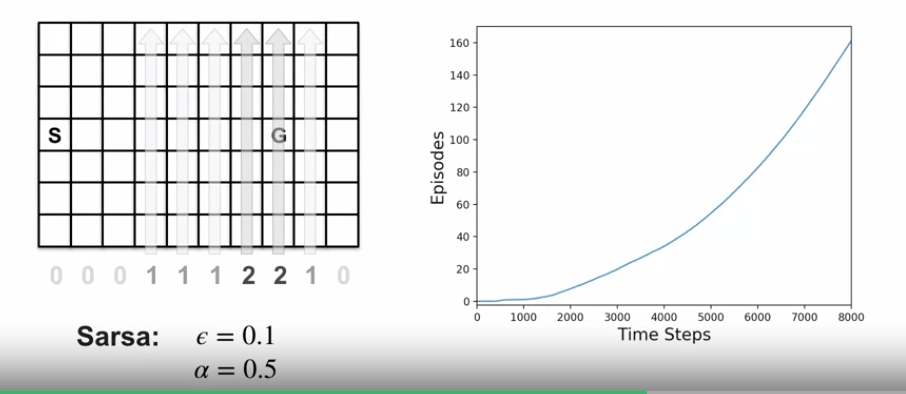
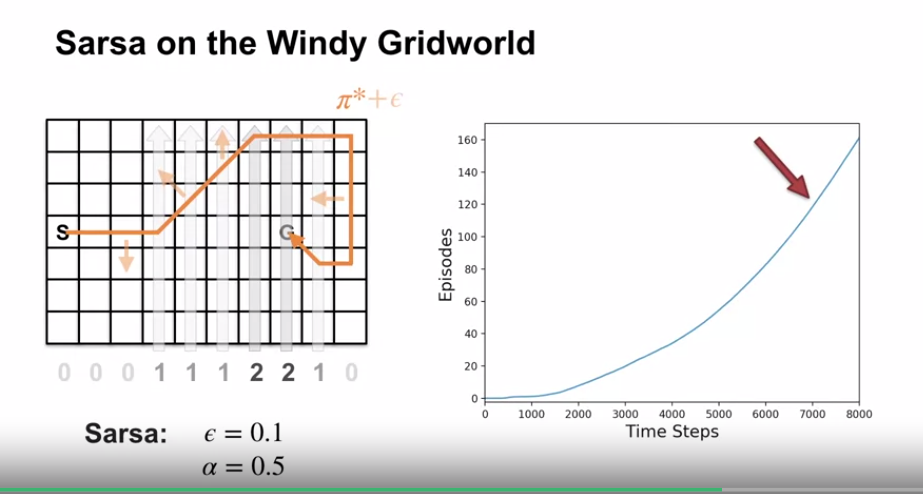
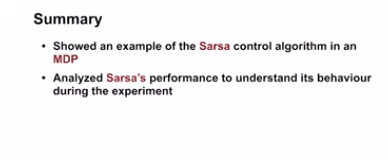
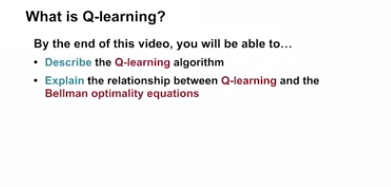
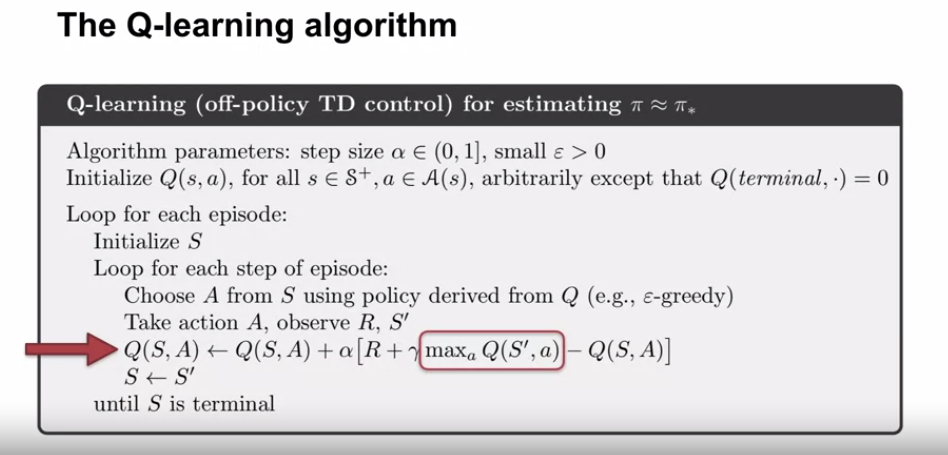
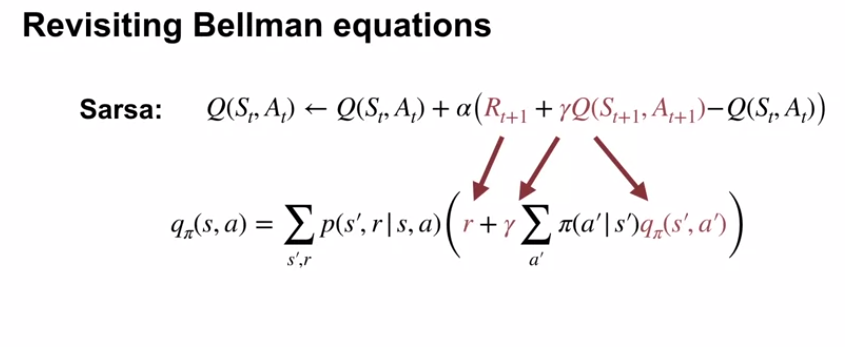
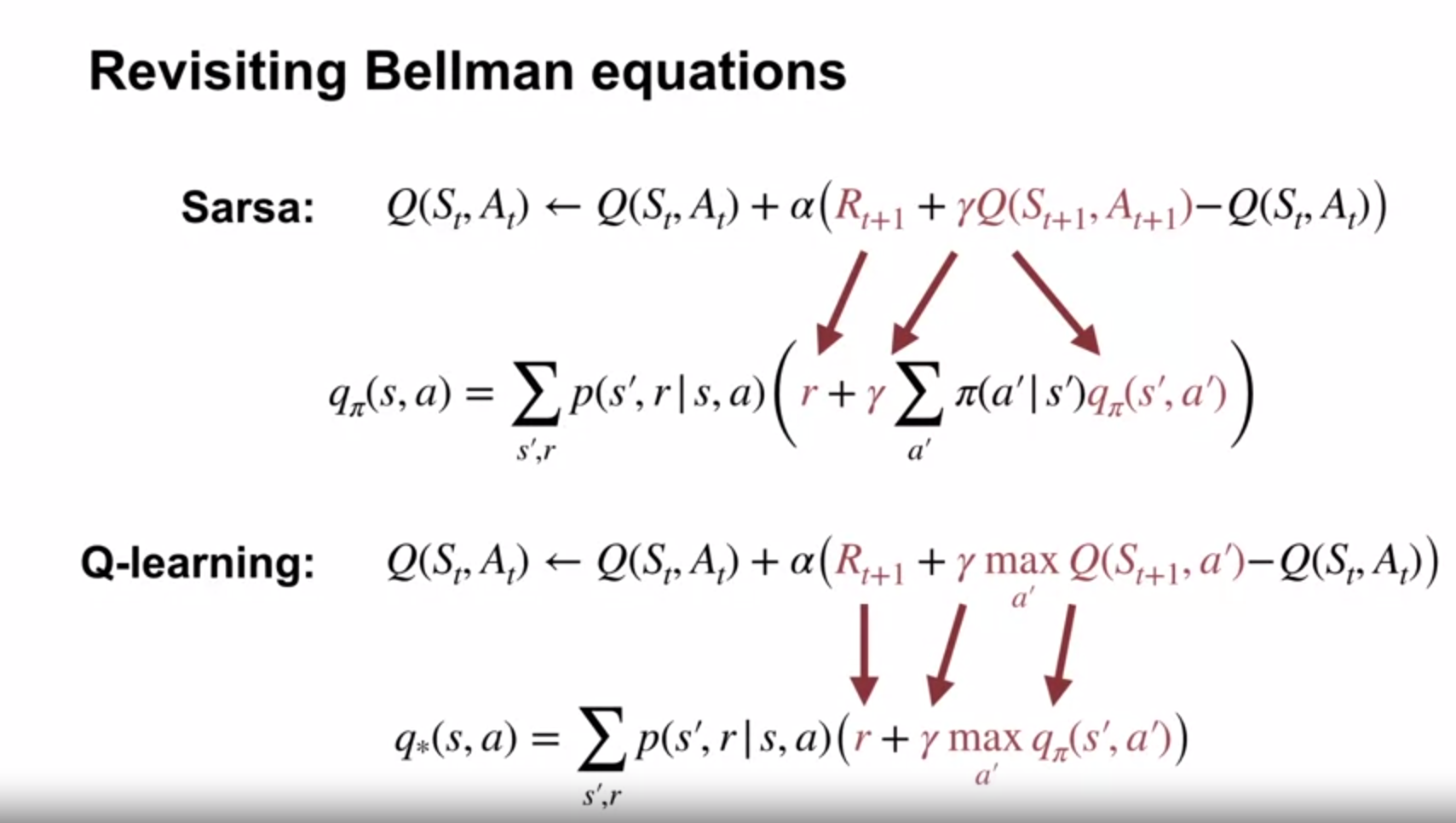
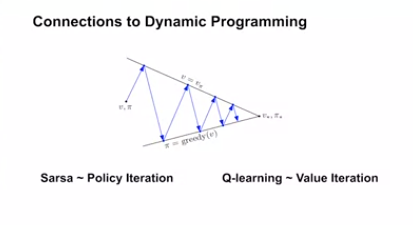
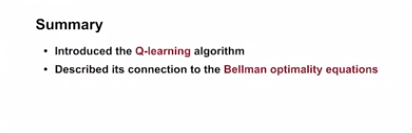
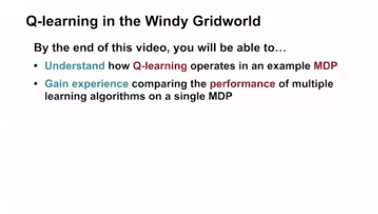
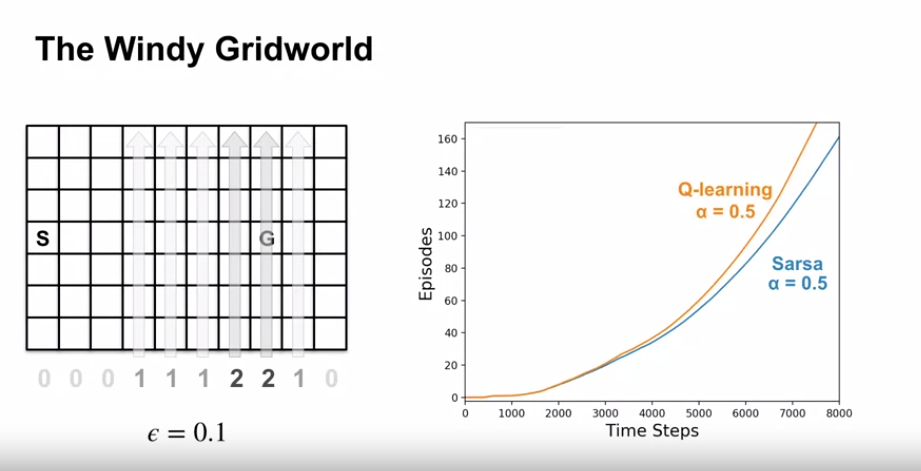
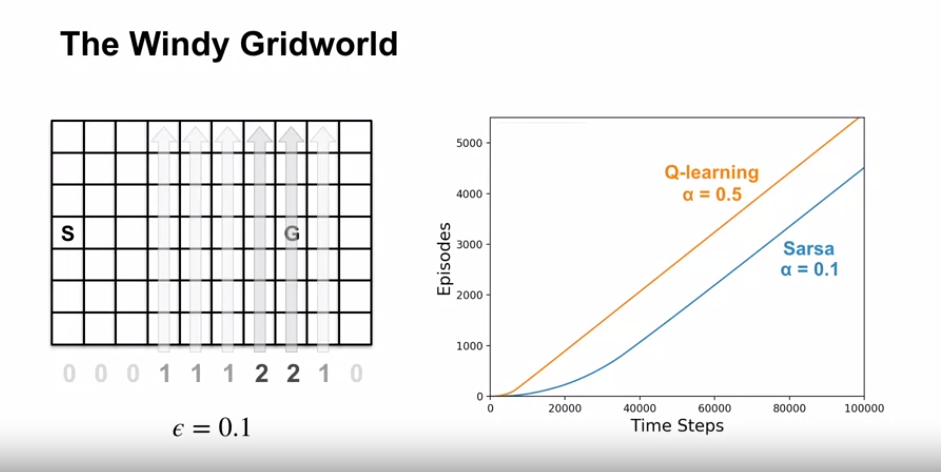
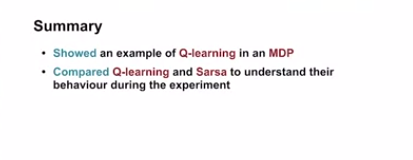
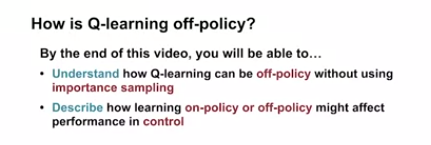
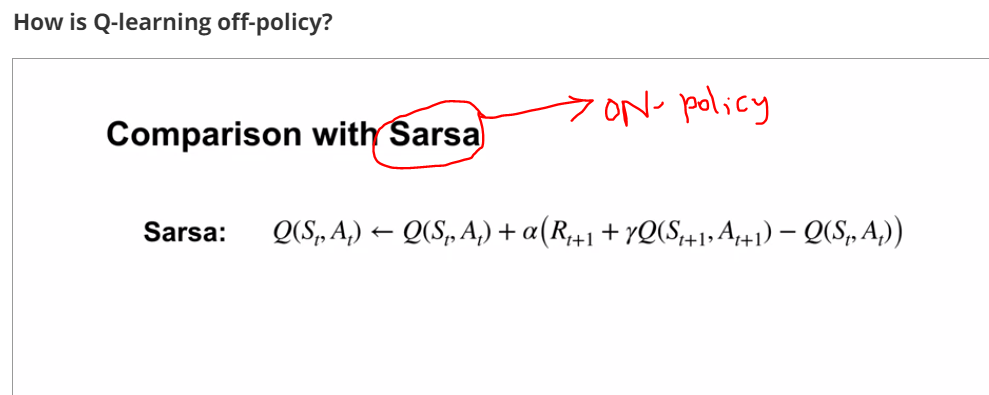
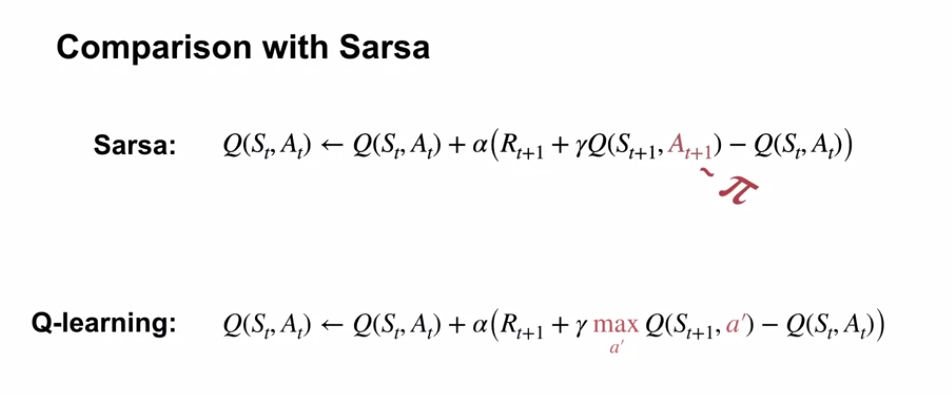
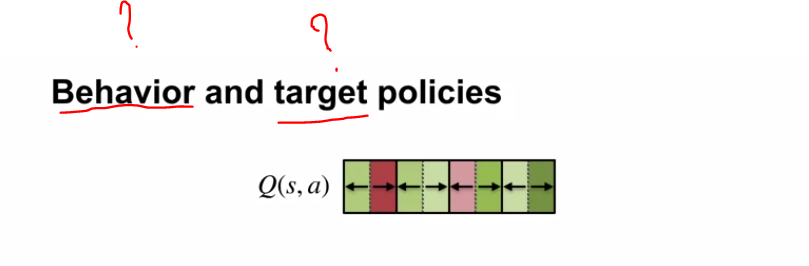
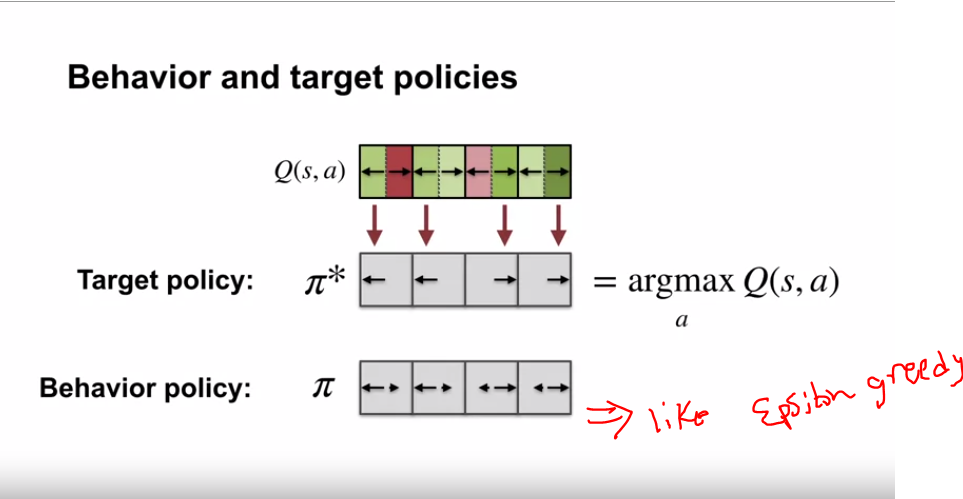
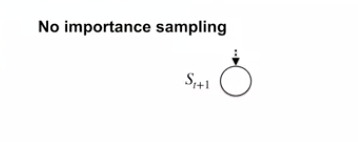
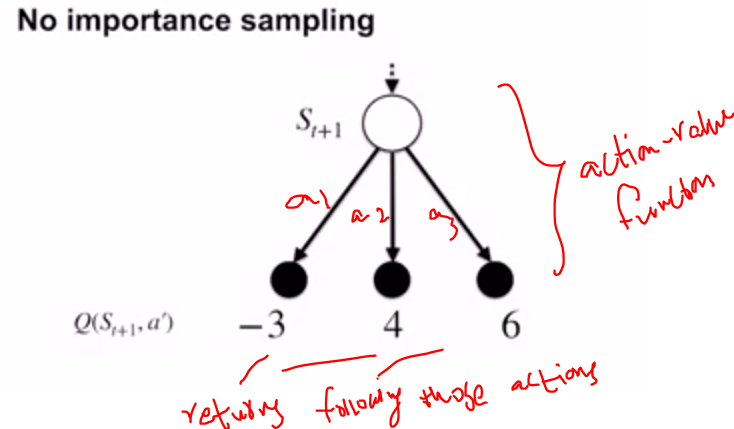
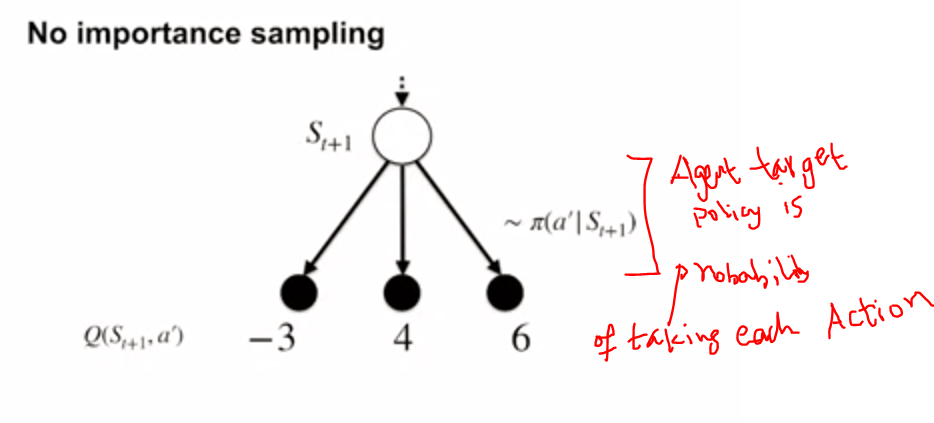
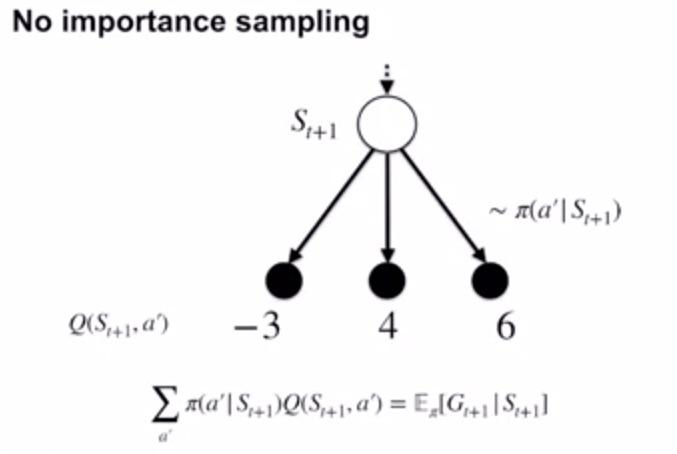
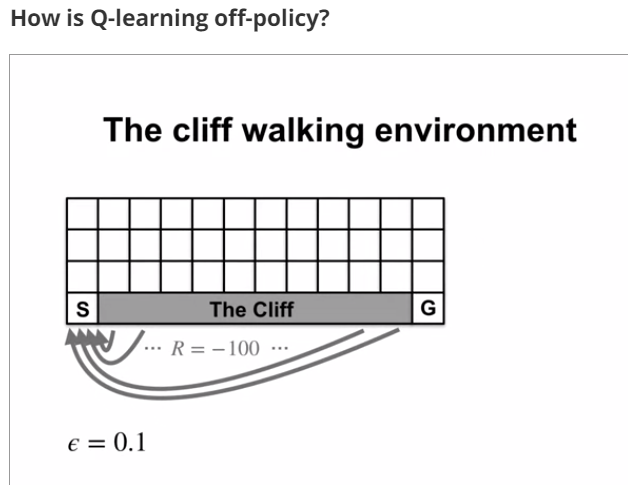
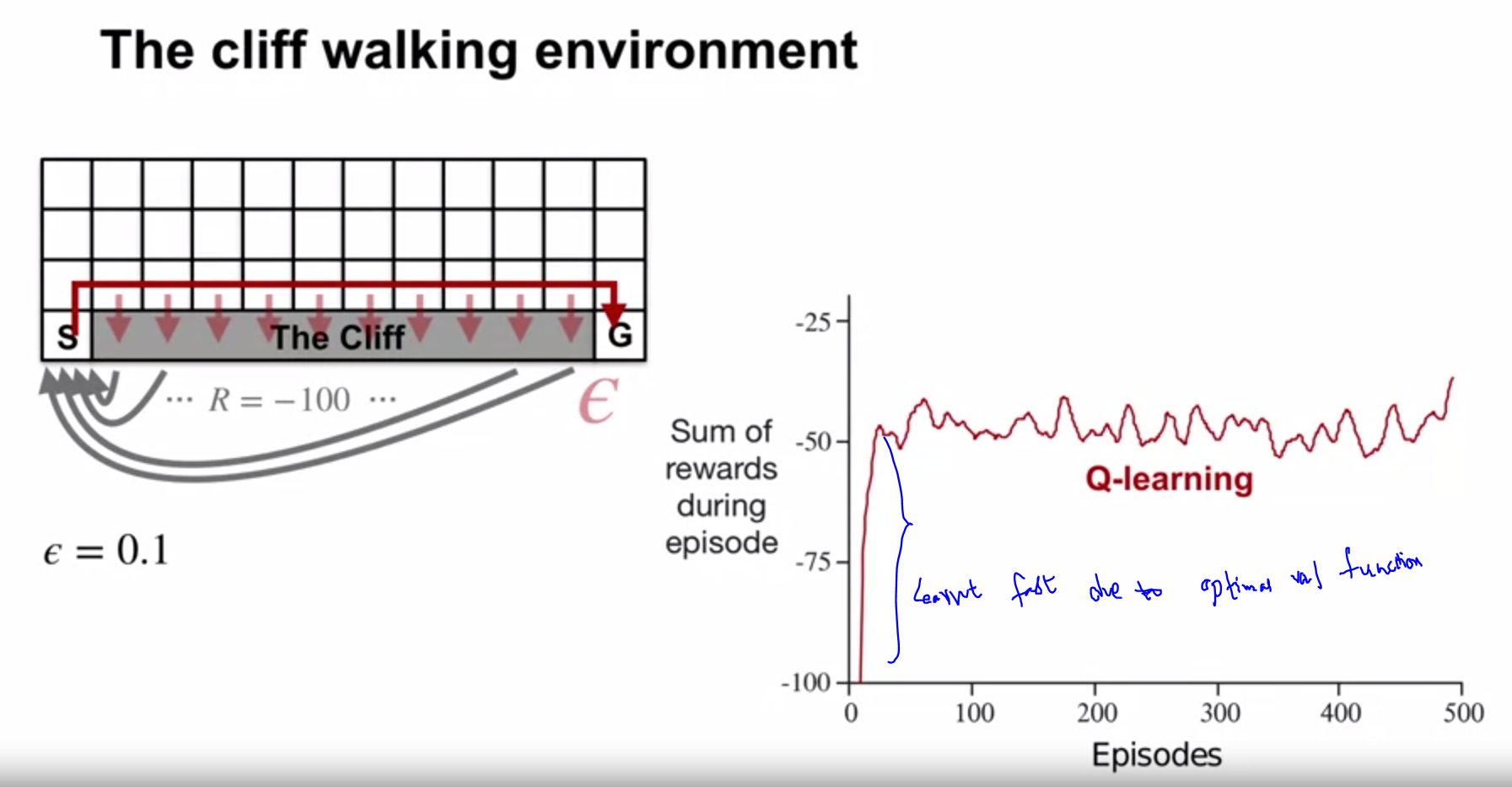
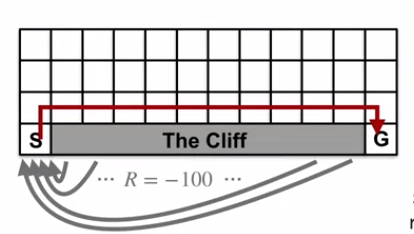
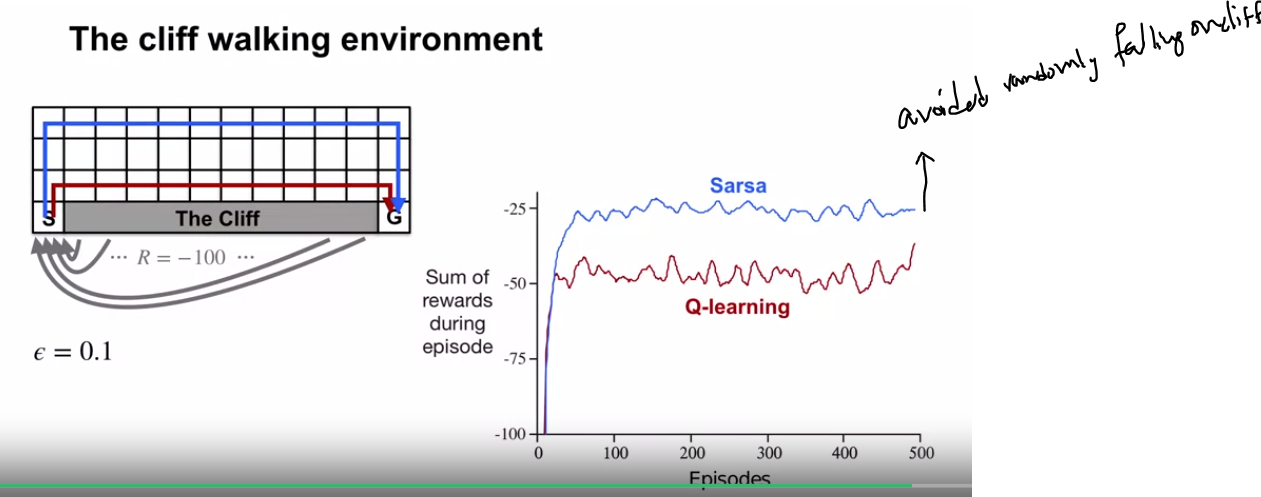
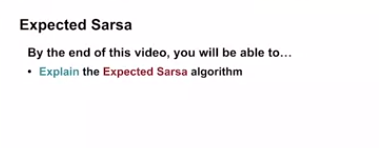
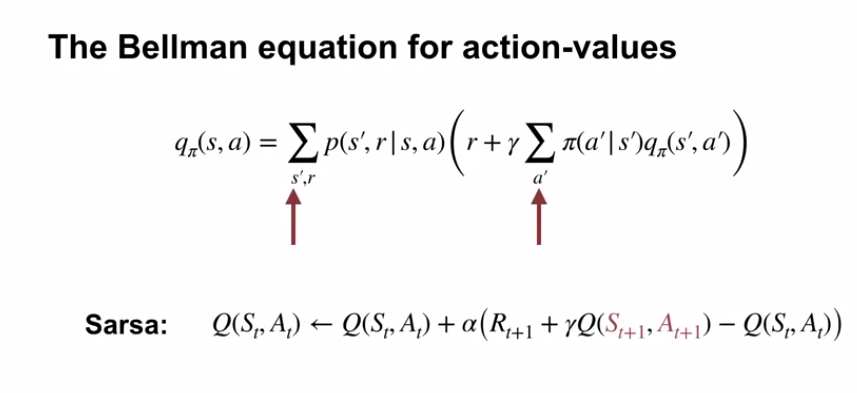
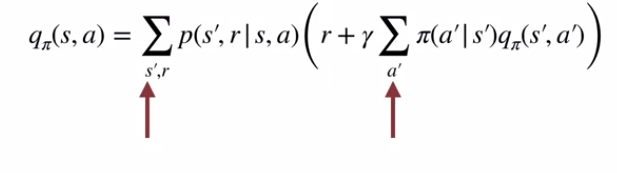
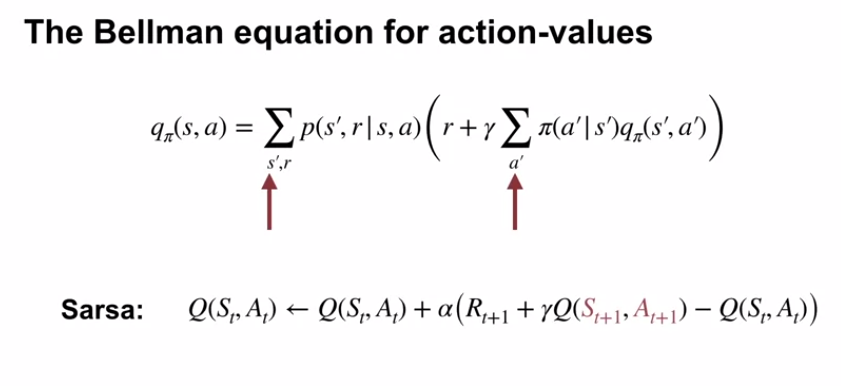
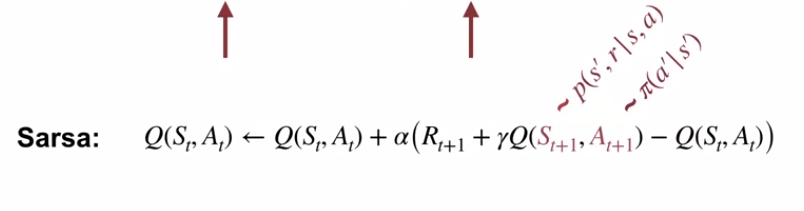
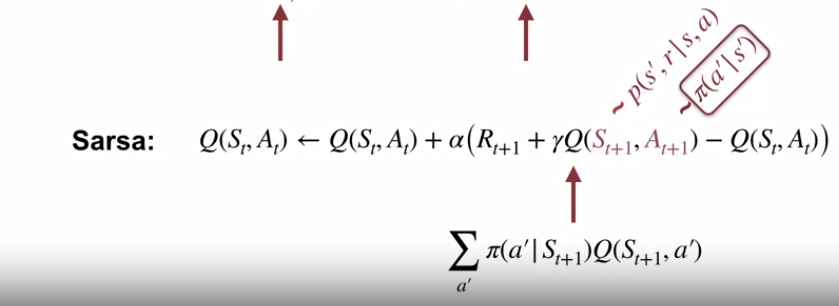
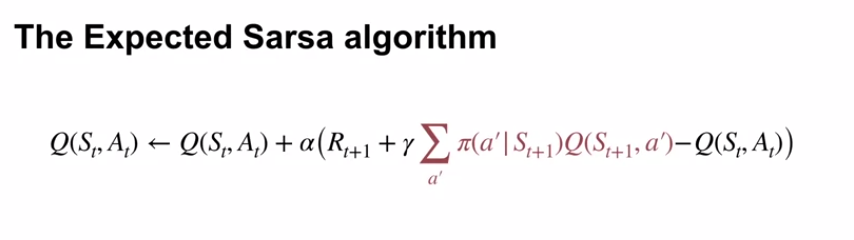
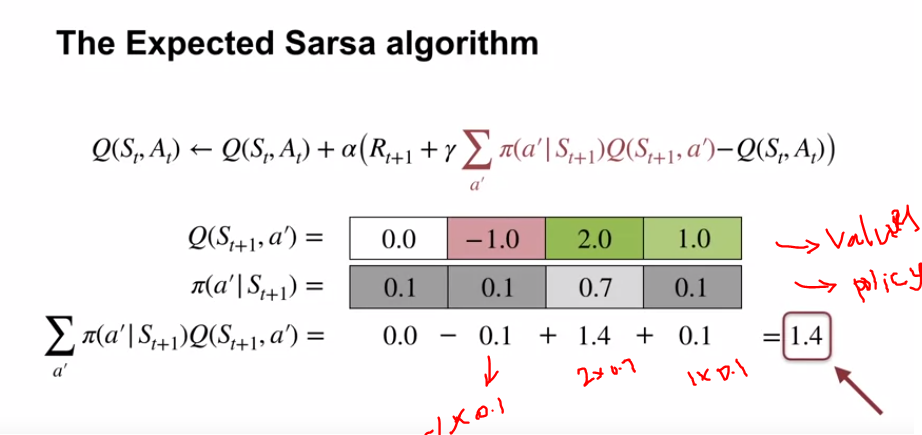
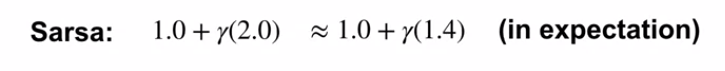
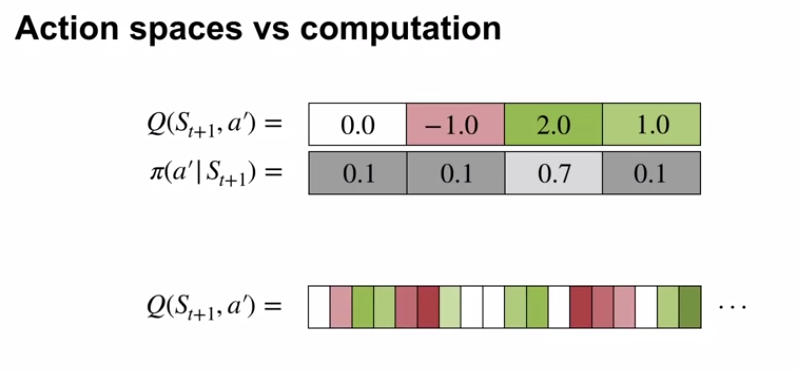
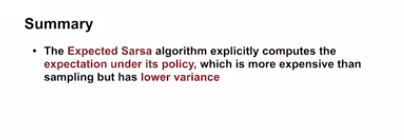
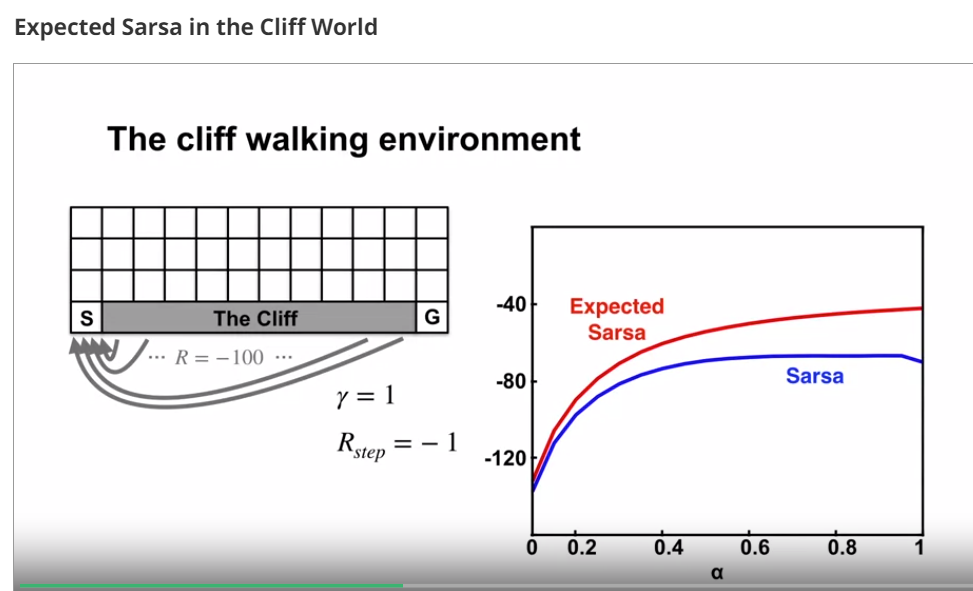
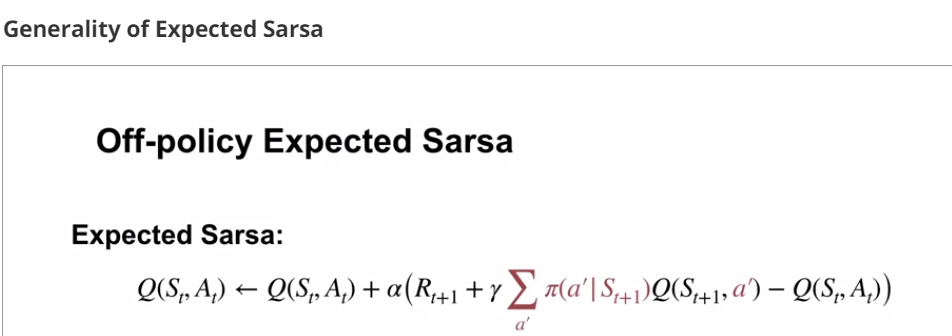
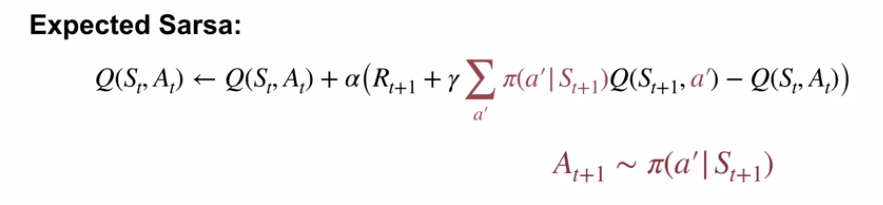
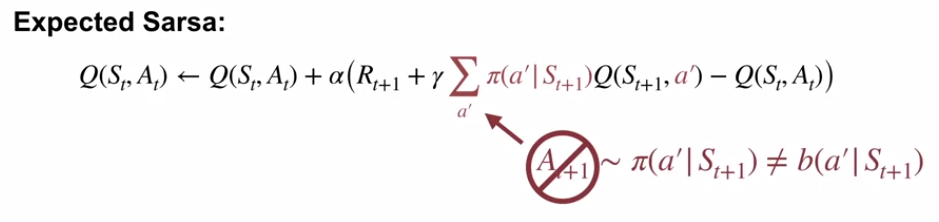
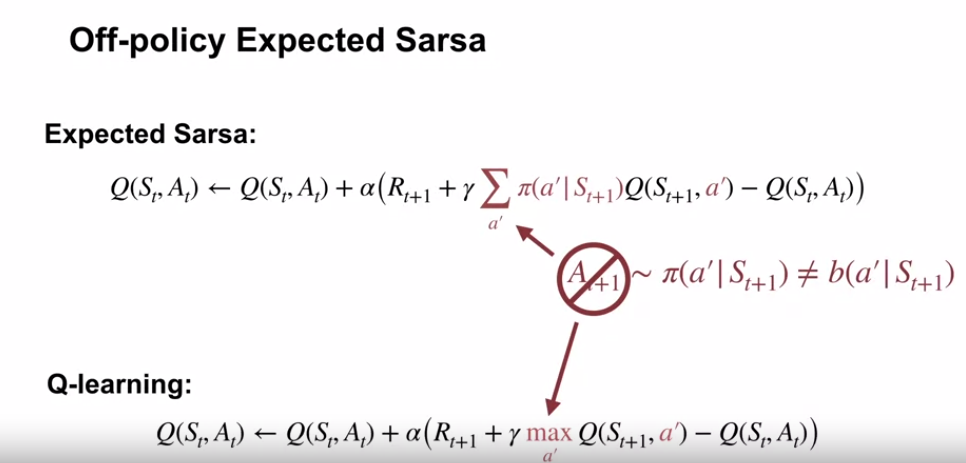
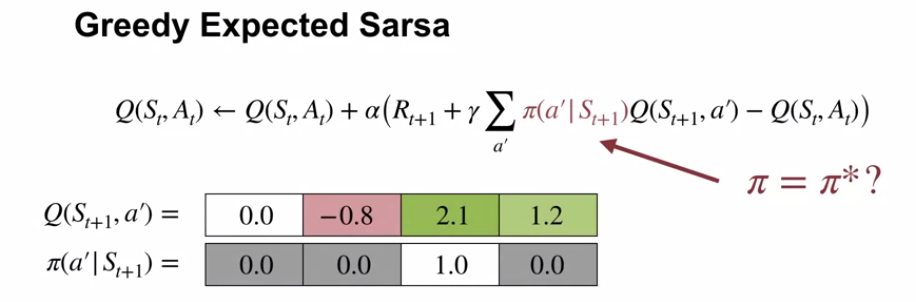
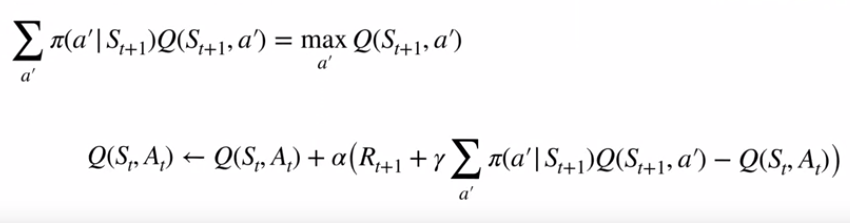
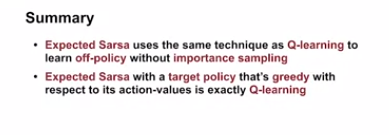
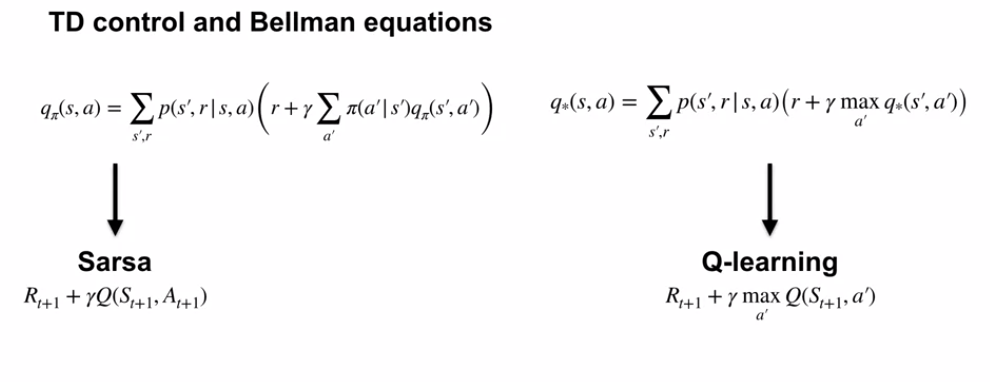
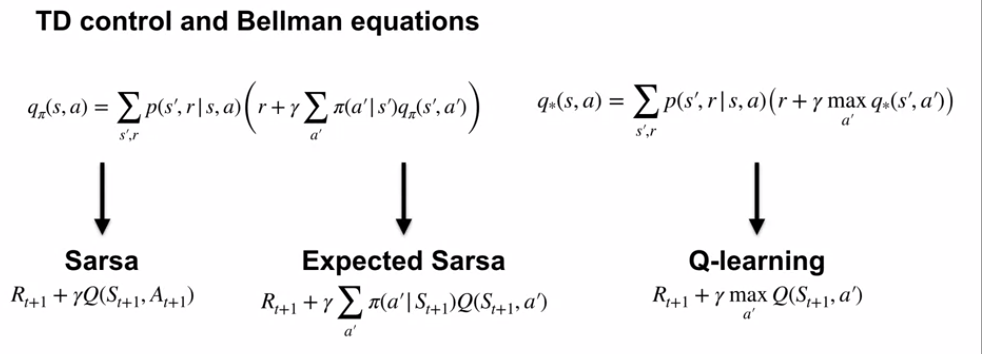
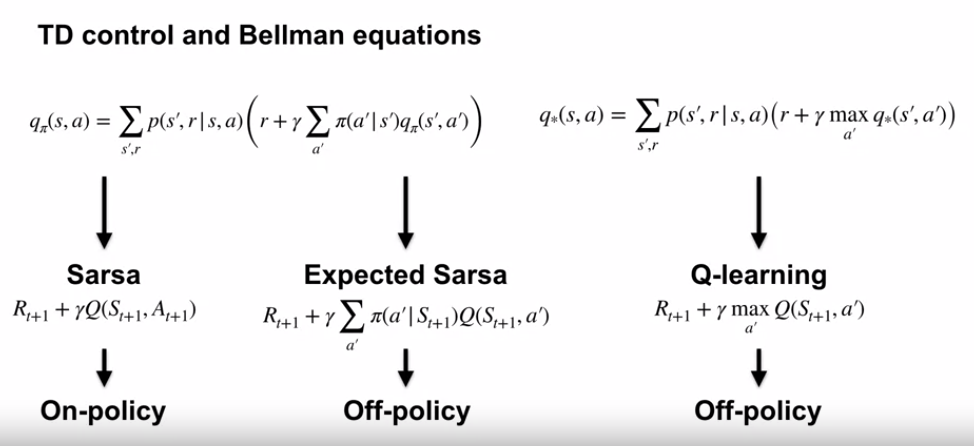
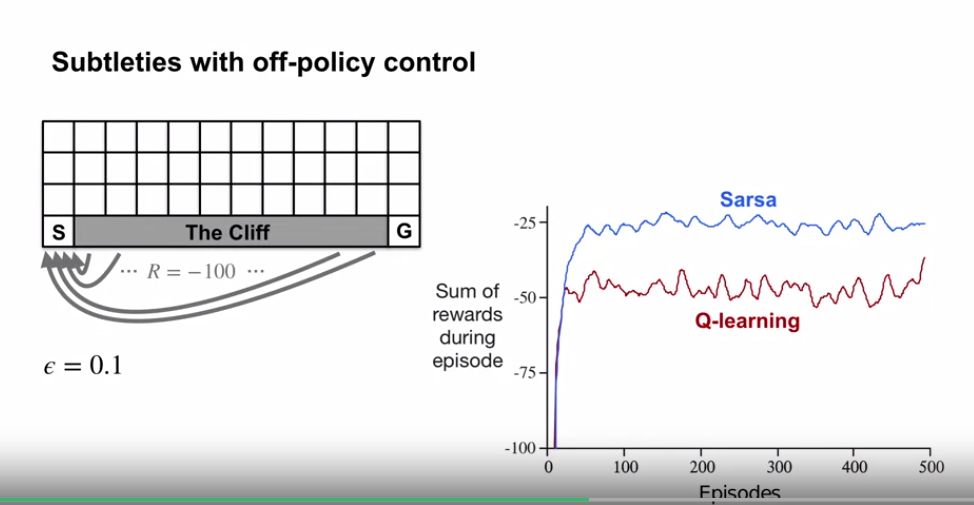
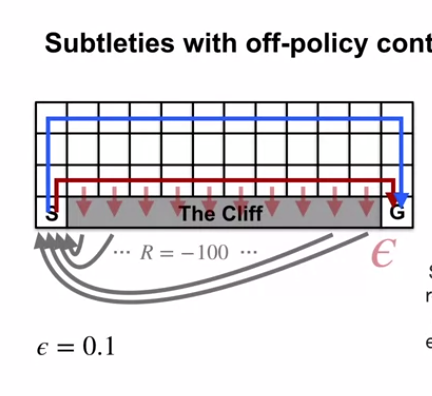
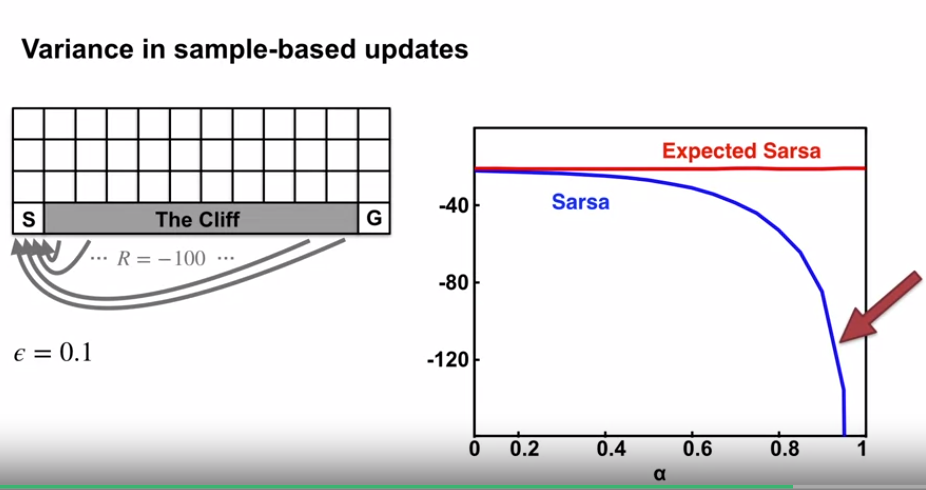
1. 
2. .
3. 
4. Above one is Monet carlo recall
5. 
6. .
7. .
8. 
9. .
10. .
11. As process repaeats it will eventually learn optimal policy
12. .
13. 
14. .
15. .
16. Note:Now lets look at GPI with TD
17. .
18. To Use GPI with TD we need to learn action-value function
19. .
20. 
21. .
22. .
23. Instead of learning from state- to state ,Now we learn **from state-action to state-action** jump
24. .
25. .
26. 
27. .
28. .
29. This is called SARSA prediction.
30. .
31. 
32. .
33. .
34. Lets say agent chooses an action At in initial state St
35. .
36. 
37. .
38. .
39. So after action At ,agent observes reward rt+1 and next state t+1
40. .
41. .
42. 
43. .
44. .
45. In **Sarsa**, the agent needs to know its **next state action pair before updating its value estimates**.
46. That means it has to commit to its next action **before** the update.
47. .
48. Since our agent is learning **action values for a specific policy**, it uses that **policy to sample the next action.**
49. .
50. .
51. 
52. .
53. .
54. 
55. .
56. .
57. 
58. .
59. .
60. The algorithm we just described is for **policy evaluation**. It learns action values for a specific **fixed policy.**
61. However, thanks the GPI framework, we can turn it into a control algorithm.
62. .
63. This time, we'll improve the policy **every time step** rather than after an episode or after convergence.
64. This completes the description of Sarsa, the GPI algorithm that uses TD for policy evaluation.
65. .
66. .
67. .
68. 
69. .
70. .
71. .
72. 
73. .
74. .
75. .
76. 
77. .
78. Reward is -1 per step .so this motivates the agent to escape punishments and reach goal as fast as possible.
79. Gamma is 1 since its episodic task
80. .
81. .
82. 
83. .
84. .
85. Since wind is blowing Upwards.when agent is met with wind it is pushed diagonney in to upper grid cells.
86. .
87. Wind strengths range is 0,1,2
88. .
89. .
90. 
91. .
92. .
93. .
94. 
95. .
96. .
97. Epsilon greedy algorithms we will apply and see
98. .sd
99. Espsilon=0.1 means 1 random action in 10 steps
100. .
101. .
102. 
103. .
104. .
105. .
106. 
107. .
108. .
109. .
110. Notice that the first few episodes take a couple thousand steps to complete. The curve gradually gets steeper indicating that episodes are completed more quickly. Around 7,000 steps, the greedy policy stops improving.
111. .
112. .
113. .
114. Notice the episode **completion rate stops increasing**. This means the agents policy **hovers around the optimal policy and won't be exactly optimal**, because of exploration. Notice that Monte Carlo methods would not be a great fit here. This is because many policies do not lead to termination.
115. Monte Carlo methods only learn when an **episode terminates**. So a deterministic policy might **get trapped and never learn a good policy in this grid world.** For example, if the policy took the left action in the start state, it would never terminate.
116. .
117. .
118. Sarsa avoid this trap, because it would learn such policies or bad during the episode. So it's switch to another policy and not get stuck..
119. .
120. .
121. 

**What is Q-learning?**

1. 
2. .
3. .
4. 
5. .
6. .
7. This differs from Sarsa, which uses the value of the next state action pair in its target.whereas here it uses the maximum action value in the following state.
8. .
9. .
10. 
11. .
12. .
13. 
14. .
15. .
16. Qlearning using bellman optimality equations for finding action values
17. .
18. .
19. The optimality equations enable Q-learning to directly learn Q-star instead of switching between policy improvement and policy evaluation steps
20. .
21. .
22. .
23. Even though Sarsa and Q-learning are both based on Bellman equations, they're based on very different Bellman equations.
24. .
25. .
26. .
27. Sarsa is sample-based version of policy iteration which uses Bellman equations for action values, that each depend on a fixed policy.
28. .
29. .
30. Q-learning is a sample-based version of value iteration which iteratively applies the Bellman optimality equation.
31. .
32. .
33. Applying the Bellman's Optimality Equation strictly improves the value function, unless it is already optimal. So value iteration continually improves as value function estimate, which eventually converges to the optimal solution.
34. .
35. .
36. For the same reason, Q-learning also converges to the optimal value function as long as the aging continues to explore and samples all areas of the state action space.
37. .
38. .
39. 
40. .
41. .
42. .
43. 
44. .
45. .
46. .
47. .
48. .
49. 
50. .
51. .
52. 
53. .
54. .
55. 
56. .
57. .
58. .
59. Lets see Q learning to same problem
60. .
61. .
62. 
63. .
64. .
65. .
66. In beginning both learn same pace, but end Q learn more good
67. .
68. .
69. Q learning strategy: Learn best action that maximizes at every state else it won’t update
70. .
71. .
72. Q-Learning takes the max over next action values. So it only changes when the agent learns that one action is better than another
73. .
74. .
75. . In contrast, SARSA uses the estimate of the next action value in its target. This changes every time the agent takes an exploratory action
76. .
77. .
78. . **How can we make SARSA perform better**? A step size parameter value of 0.5 might be a bit high for this experiment. These large updates might be causing SARSA trouble when the agent takes exploratory actions. SARSA'S final policy certainly isn't optimal. Let's run the experiment a bit longer, and this time with Alpha equal 0.01. We expect SARSA to learn more slowly with the smaller alpha but find a better policy. Let's see what happens. Prediction confirmed.
79. .
80. .
81. .
82. 
83. .
84. .
85. .
86. Same result sarsa performed badly than Q-learning
87. .
88. .
89. .
90. .
91. SARSA learns the same final policy as Q-Learning, but more slowly. We know both algorithms have to converged to the same policy because the slopes of the lines are equal.
92. Equal slopes mean that both agents are completing episodes at the same rate.
93. .
94. .
95. .
96. This experiment also highlights the impact of parameter choices in reinforcement learning. Alpha, Epsilon, initial values, and the length of the experiment can all influence the final result.
97. .
98. .
99. .
100. .
101. 
102. .
103. .
104. .
105. 
106. .
107. .
108. Q-Learning is an off-policy algorithm. But so far, we've only seen off-policy algorithms that use important sampling. How can Q-learning be off policy without using important sampling?
109. .
110. .
111. .
112. Recall that an agent estimates its value function according to expected returns under their target policy. They actually behave according to their behavior policy. When the target policy and behavior policy are the same, the agent is learning on-policy, otherwise, the agent is learning off-policy.
113. .
114. .
115. 
116. .
117. .
118. .
119. 
120. .
121. .
122. In Sarsa, the agent bootstraps off of the value of the action it's going to take next, which is sampled from its behavior policy.
123. .
124. .
125. .
126. Q-learning instead, bootstraps off of the largest action value in its next state. This is like sampling an action under an estimate of the optimal policy rather than the behavior policy.
127. .
128. .
129. .
130. Since Q-learning learns about the best action it could possibly take rather than the actions it actually takes, it is learning off-policy.
131. .
132. .
133. .
134. .
135. 
136. .
137. .
138. .
139. 
140. .
141. .
142. .
143. .
144. .
145. .
146. 
147. .
148. .
149. .
150. .
151. .
152. 
153. .
154. .
155. .
156. The difference here between the target and behavior policies confirms that Q-learning is off-policy.
157. .
158. .
159. . 
160. .
161. .
162. .
163. if Q-learning learns off-policy, why don't we see any important sampling ratios?
164. sol: It is because the agent is estimating action values with unknown policy. It does not need important sampling ratios to correct for the difference in action selection.
165. .
166. .
167. .
168. .
169. The action value function represents the returns following each action in a given state.
170. .
171. .
172. . 
173. The agents target policy represents the probability of taking each action in a given state.
174. .
175. .
176. 
177. .
178. Putting these two elements together, the agent can calculate the expected return under its target policy from any given state, in particular, the next state, S\_t plus 1.
179. .
180. .
181. .
182. . 
183. .
184. . Since the agents target policies greedy, with respect to its action values, all non-maximum actions have probability 0.
185. As a result, the expected return from that state is equal to a maximal action value from that state
186. .
187. .
188. .
189. 
190. .
191. .
192. Compare sarsa vs Q-learning
193. .
194. .
195. .
196. Since Q-learning learns the optimal value function, it quickly learns that an optimal policy travels right alongside the cliff.
197. .
198. .
199. 
200. .
201. However, since his actions or epsilon greedy, traveling alongside the cliff occasionally results and falling off of the cliff.
202. .
203. 
204. .
205. .
206. .
207. Sarsa learns about his current policy, taking into account the effect of epsilon greedy action selection.
208. Accounting for occasional exploratory actions, it learns to take the longer but more reliable path. They usually avoids randomly falling into the cliff. Because of it's safer path, Sarsa is able to reach the goal more reliably.
209. .
210. .
211. 
212. .
213. .
214. .
215. In this video, we showed that Q-learning is off-policy without using important sampling, and that learning on-policy or off-policy may perform differently in control depending on the task.
216. .
217. .
218. .
219. 
220. .
221. .
222. .
223. .
224. .
225. .
226. 
227. .
228. .
229. .
230. 
231. .
232. .
233. Recall the Bellman equation for action-values. Here you can see the expectation over values of possible next state action pairs.see below
234. .
235. 
236. Breaking this expectation apart, we see a sum over possible next states as well as possible next action.like below
237. .
238. .
239. 
240. .
241. .
242. .
243. Sarsa estimates this expectation by sampling the next state from the environment and the next action from its policy.see below
244. .
245. .
246. . 
247. But the agent already knows this policy, so why should it have to sample its next action? Instead, it should just compute the expectation directly.
248. .
249. .
250. In this case, we can take a weighted sum of the values of all possible next actions. The weights are the probability of taking each action under the agents policy.
251. .
252. .
253. .
254. 
255. .
256. Explicitly computing the expectation over next actions is the main idea behind the expected Sarsa algorithm.
257. .
258. .
259. Explicitly computing the expectation over next actions is the main idea behind the expected Sarsa algorithm.
260. .
261. .
262. The algorithm is nearly identical to Sarsa, except the T error uses the expected estimate of the next action value instead of a sample of the next action value. That means that on every time step, the agent has to average the next state's action values according to how likely they are under the policy.see below
263. .
264. .
265. .
266. 
267. .
268. .
269. 
270. .
271. .
272. .
273. For example, with the following values and policy, expected Sarsa would use a value of 1.4 for its estimate of the expected next action value.
274. However, there's a huge upside to calculating the expectation explicitly.
275. Expected Sarsa has a more stable update target than Sarsa.
276. .
277. .
278. .
279. Let's look at an example to make this more clear. In this example, the immediate reward is deterministically 1.
280. .
281. .
282. .
283. .
284. Both Sarsa and expected Sarsa, start up with a true action values for the next state.
285. .
286. .
287. Even in this idealized case, the next action sampling that Sarsa does can cause it to update its values in the wrong direction.
288. .
289. .wrong direction like 
290. . 
291. 
292. . 
293. .
294. .
295. .
296. . 
297. It relies on the fact that in expectation across multiple updates, the direction is correct.
298. .
299. . 
300. .
301. By contrast, expected Sarsas update targets are exactly correct, and do not change their estimated values away from the true values.
302. .
303. .
304. .
305. 
306. .
307. In general, expected Sarsas update targets are much lower variance than Sarsas.
308. .
309. .
310. The lower variance comes with a downside though. Computing the average over next actions becomes more expensive as the number of actions increases.
311. .
312. .
313. 
314. .
315. .
316. When there are many actions, computing the average might take a long time, especially since the average has to be computed every time step.
317. .
318. .
319. In this video, we show that the expected Sarsa algorithm explicitly computes the expectation under its policy, which is more expensive than sampling but has lower variance.
320. .
321. .
322. 
323. .
324. .
325. .
326. 
327. .
328. . Sarsa performed better than Q learning on this domain because Sarsa's policy accounted for its own exploration.
329. .
330. .
331. Today, we'll compare expected Sarsa to Sarsa with an epsilon of 0.1. This plot was created by testing each agent with different values of the step size parameter alpha.
332. 
333. .
334. .
335. .
336. We did a hundred episodes and averaged everything over 50,000 independent runs.
337. .
338. .
339. The y axis shows the average return per episode and the x axis shows the step size values. Sarsa's performance increases with larger step size values, but only up to a point.
340. .
341. The best alpha value for Sarsa here was 0.9. Expected Sarsa outperformed Sarsa for almost all values of alpha.
342. .
343. .
344. Expected Sarsa's able to use larger alpha values more effectively. This is because it explicitly averages over the randomness due to its own policy. This environment is deterministic, so there are no other sources of randomness to account for.
345. .
346. .
347. .
348.  detreminsitic
349. .
350. .
351. This means expected Sarsa's updates are deterministic for a given state and action. Sarsa's updates on the other hand can vary significantly depending on the next action.
352. .
353. .If more determentistic means avoids wrong direction of juggling going back and front
354. .
355. .
356. Now let's look at the average return per episode after 100,000 episodes. At this point, each algorithm has learned everything it's going to learn.
357. Expected Sarsa's long-term behavior is unaffected by alpha. Its updates are deterministic in this example. Therefore the step size only determines how quickly the estimates approach their target values.
358. Sarsa behaves quite differently here, it even fails to converge for larger values of alpha. As alpha decreases, Sarsa's long run performance approaches expected Sarsa's.
359. .
360. .
361. In this video, we've seen that expected Sarsa was able to quickly learn a good policy in the cliff world and that expected Sarsa is more robust than Sarsa to large step sizes
362. .
363. .
364. Two of those algorithms, Sarsa and Expected Sarsa both approximate the same Bellman equation. Today, let's look at how Expected Sarsa is related to Q-learning.
365. .
366. .
367. .
368. 
369. .
370. .
371. . Let's start with the on-policy case, where the behavior policy and the target policy are equal. Consider the Expected Sarsa update.see below
372. .
373. . 
374. .
375. .
376. .
377. The next action is sampled from Pi in this case.
378. .
379. . 
380. .
381. However, notice that the expectation over actions is computed independently of the action actually selected in the next state.
382. .
383. . 
384. .pi=target policy here
385. In fact, Pi need not be equal to the behavior policy. This means that Expected Sarsa, like Q-learning, can be used to learn off-policy without importance sampling.
386. .
387. . 
388. .
389. .
390. . 
391. .
392. Now, what happens if the target policy is greedy with respect to it's action value estimates?
393. ..
394. .
395. 
396. .
397. We can see that only the highest value action is considered in the expectation. This is equivalent to computing the maximum over actions in the next state just like in Q-learning.
398. .
399. .
400. .
401. In other words, Q-Learning is a special case of Expected Sarsa.
402. .
403. .
404. . 
405. .
406. . 
407. .
408. In other words Q-learning is special case of Expected sarsa
409. .
410. In this video, we showed that expected Sarsa and Q-Learning both use the expectation over their target policies in their update targets. This allows them to learn off-policy without importance sampling. Expected Sarsa with the target policy that's greedy with respect to its action values, is exactly Q-learning.
411. .
412. .
413. .
414. 
415. .
416. .
417. .
418. The TD control algorithms are based on Bellman equations
419. .
420. .
421. 
422. .
423. .
424. We learned about three of them. Sarsa uses a sample based version of the Bellman equation. It learns Q-pi.
425. Q-learning uses the Bellman optimality equation. It learns Q-star.
426. .
427. 
428. .
429. .
430. Expected sarsa uses the same Bellman equation as Sarsa, but samples it differently. It **takes an expectation over the next action values**.
431. .
432. .
433. .
434. . 
435. .
436. .
437. What's the story with on-policy and off-policy learning? Sarsa is a on-policy algorithm that learns the action values for the policy it's currently following. Q-learning is an off-policy algorithm that learns the optimal action values.
438. And Expected Sarsa is both an on-policy and an off-policy algorithm that can learn the **action values for any policy**.
439. .
440. .
441. .
442. 
443. .
444. .
445. .
446. ..
447. .
448. Sarsa can do better than Q-learning when performance is measured online.
449. This is because on-policy control methods account for their own exploration.
450. .
451. .
452. . 
453. .
454. .
455. In the cliff world we saw that q-learning frequently fell off the cliff because of its **exploratory** actions. Sarsa learned the longer but safer path that rarely fell off the cliff, this resulted in higher reward.
456. .
457. .
458. . 
459. .
460. .
461. We then studied an improvement over Sarsa called Expected Sarsa.
462. In the cliff world Expected Sarsa outperformed Sarsa for all the step size parameter values we tested.
463. .
464. .
465. .
466. This is because Expected Sarsa mitigates the variance due to its own policy. Expected Sarsa, like the name suggests, takes the expectation over the next action.
467. . 
468. .
469. .
470. .