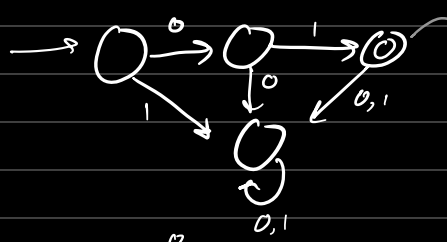
Reinforcement Learning

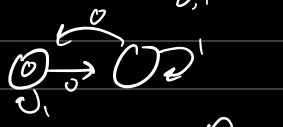
Finite State Machine / Deterministic Finite Automata

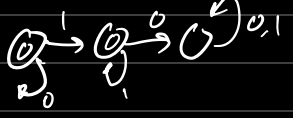
- Acceptors / recognizer → returns T/F based on final state in graph



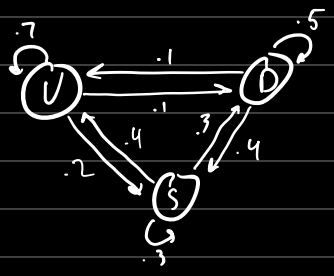
01 is only accepted string

- can encode regex like $(01)^+$ or $(01)^*$

ex a)  even # of 0's

b)  any # of 0's then any # of 1's → 0^*1^*

Markov Chain



encodes FJJ

$P(S'=D|S=D) = .5$

$P(S'=U|S=S) = P(UU) + P(SU) + P(DU)$
$t+2 \qquad = .4(.7) + .3(.4) + .3(.1)$

Adding inputs and Actions → MDP need tuples S, a, S', R

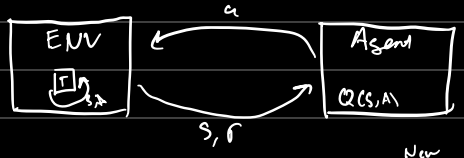
$T(S,a,S') = P(S'|S,a)$ ← use reinforcement learning to learn this

$R(S)$ $R(S,A)$ $R(S,a,S')$

$V(S)$ ~ goodness of state

$Q(S,a)$ = total expected rewards for a discounted

$\Pi(S) = \arg\max_a Q(S,a)$



old — new: reward + expected future rewards

$$Q'[S,a] = (1-\alpha)\,Q[S,a] + \alpha\,(r + \gamma \max_{a'} Q(S',a'))$$

γ: discount rate

exploration exploitation → epsilon greedy / decay
rar radr

for project will train and test same except test does not learn and take no random actions

Alternative to epsilon greedy

- epsilon greedy ignores actions that are close to the best $Q_s' = \begin{cases} .49 \\ .48 \\ .02 \\ .01 \end{cases}$
  - Boltzmann (softmax): $P(a|s) = \frac{e^{Q_{sa}}}{\sum_j e^{Q_{sj}}}$ "normalized exponential"

Problems w/ QL

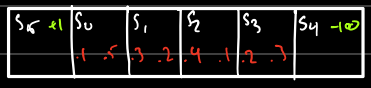
1. discreet state-action space
   - ↳ never enough training data for large grid → Can artificially generate more samples for training
   - make the table continuous: Deep Q
   - $Q[s,a] \rightarrow \underset{\text{network}}{Q(s,a)}$ neural network
2. Why did I get this reward?
   - ↳ sparse reward spread out over lots of actions
   - ↳ chess ex: 100 moves → win or loss what caused it?
   - ↳ credit assignment problem
3. Experimentation is SLOW!
   - ↳ ex stocks / cars only get data from trying → slow
4. Experimentation is Costly
   - ↳ risk real $ or damage
   - ↳ train in simulation first

Ex

$\alpha = .3$

$\gamma = .9$

Steep slope to cliff where $S_1, S_2, S_3, S_4 \rightarrow$ cliff



Go Right → $\langle S_0, R, S, 0 \rangle$

$Q[S_0,R] = .7(.5) + .3(.09(.03)) = .431$

Try left → $Q[S_1,L] = .318$

Try left → $Q[S_2,L] = .361$

Try Right → $Q[S_3,R] = -29.79$

Going to take lots of iterations to figure out that all of actions beginning with right are bad needs to hit cliff repeatedly and eventually error is propagated back to root cause

How to solve? → hallucinate experiences → "Dyna-Q"

Model + model free approach — Slower than Q, but reduces real world interaction

Calculate T & R

Idea:

Loop forever:

1. Q-learning iteration as usual → real experience
2. estimate T & R from experience

$T[S,A,S'] = \frac{T_c[S,a,s']}{\sum_i T_c[S,a,i]}$ → matrix of counts a transition has occurred → normalize for all adjacents from current

dreaming?

conciousness representation of state?

$R[S,A] = (1-\alpha)R[S,A] + \alpha r$ ← this experience

3. Hallucinate n times (n ← hyper parameter) : sample from model
   - $\hat{s}$ is random state we've seen before → weight by # times seen before?
   - $\hat{a}$ is random action from $\hat{s}$ seen before
   - $\hat{s}', r = model(\hat{s}, \hat{a})$
   - Q update

Alternative: Store all experience in a dict and uniformly sample from it

Q-learning (ipad was dead last class)

- Main takeaways, can make alpha depend on # of times the state has been seen

Assumptions

$r_t \sim R : S \times A \times S' \to \mathbb{R}$ (fixed function)

$\mathbb{E}(r^t | s,a,s') = R^{s'}_{sa}$

$S_{t+1} \sim P : S \times A \times S \to [0,1]$ (fixed), $\sum_{s'} P^{s'}_{sa} = 1$

↳ Probabilities

discount $\gamma \in [0,1)$: domain dependent must be < 1 or else infinite time to get reward

learning $\alpha \in (0,1]$: average over $r_t \sim R$, need to have some sort of memory

$Q' = (1-\alpha) Q[s,a] + \alpha(r + \gamma \max Q(s',a'))$

↳ overestimation bias "overly optimistic"

leads to problems at beginning of training

- how to break this cycle? 2 learners

Double Q, 2 learners, Q^A, Q^B

- outlier experience only goes to one learner
  ↳ if Q^A overestimates, Q^B underestimates

- Algorithm: Store Q^A, Q^B
  1. Select random table lets say you set A
  2. Use to get action / update $Q'^A = (1-\alpha)Q^A[s,a] + \alpha(r + \gamma Q^B(s', \operatorname{argmax}_{a'} Q^A(s',a')))$

Dependency Problem

Core ↙ Handle ↓ Delegates for defaults ← client = none

Handle ↓ inject the delegate in core?

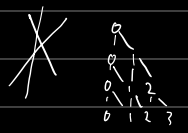
Core for delegate model ↓

model.client = self

- need twice as much data could use average of Q tables

- Roulette example, normal Q learns $22.63-22.67 after 100,000 trials
  double Q learns <0 after 200 trials

dropping lines / intersections R.V



- Problems w/ Q trader
  - not enough data: non stationary } hard to fill discreet Q table
  - need lots of trials

Less

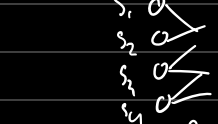
Decision Tree & Learns type?

- Work arounds
  - A. limit actions
  - B. State neighborhoods
  - C. Dyna Q: data augmentation
    - ↳ ex: for image, flip, tint, etc to get more data
    - ↳ could add noise but maintain trend → for loops?
  - D. ABIDES: simulate w/ computers / agents to get more data
    - ↳ Agent based Interactive descreet event simulation
  - E. train on multiple stocks to learn one
  - F. Make less states (bin)

- Neural Network → continuous state space $\mathbb{R} \to \mathbb{R}$

- Deep Q-Learning (DQ network)

- Q-table → Q-network



} pick highest activation

↳ features ↳ Q values

- Atari Example
  - S: image of screen raw pixels
  - partially observable
  - perceptually Aliased - moving image - don't know direction (different things map to same image)
  - Actions: 8 directions, 1 button → 18 (2 nothings)
  - Reward: score don't know function $\sum_{t=t}^{T} \gamma^{t-1} r_t$
  - Transition: don't know
  - loss: error between estimated Q and true Q: GD

$Q(s,a)$ vs. $r + \gamma \max_a (Q(s', a))$ part of Q update

↳ gradient descent has a built in "learning rate"

- Use convolutional neural network for image → is Blokus board an image that is pixelated?

$[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]^2$

- Problems
  1. not sample efficient
  2. consecutive samples is not very efficient - correlated
     - not much time for action to have consequences, screen doesn't change much
  3. Online learning drives NN to sub optimal results
     - feedback loop from initial action - determines future states

Dyna helps!

"experience replay"

dream? mix experiences - common and rare and simulate using mental model to get new experience. Model transition matrix as neural network? reward/happiness as part of state? instead of seperate?

- Experience Replay
  - store all exp in replay memory
  - after each real exp.
    - Q-updates on mini-batches of experience randomly drawn from memory
  - ϵ-greedy for "a" as normal
  - ↳ run into memory issues with so many large states
    - can use circular buffer (overwrite oldest things)
  - ↳ Sampling uniformly might not be ideal
    - weight recent stuff more heavily or more impactful, or rare/common
  - ↳ down sample reduce state data? ~~need out~~
  - ↳ can stack multiple frames into one state
  - ↳ hard to train long term storage
  - ↳ can do Boltzmann exploration w/ softmax at final layer → Π and sample

- Problem: Q learning can diverse → extremly different values as train
  - Gradient descent also can hit local optima
  - ∞ state space invalidates the optimality proofs / guarantees
  - also some problem as double Q → overestimation bias
    - ↳ double deep Q learning
      - training 2 is really expensive

1. Q-network: selects action and gets update
2. target network: periodic slow copy of Q → no training or action selection (eval only)
   - ↳ doesn't contain recent experiences - batch

$$[r + \gamma \max_{a'} Q^T(s',a') - Q(s,a)]^2$$

↳ not fully decoupled as same and value are from same network

$$[r + \gamma Q^T(s', \text{argmax}_{a'} Q(s',a')) - Q(s,a)]^2$$

↳ also ask Q network what it would do next → making all the actions

- Coach learner? learns how to alter rewards from state to optimize learning

- Advantage learning
  - actor-critic algorithms: 2 distinct learning entities
  - Critic evaluation: advantage criterion - better or worse than expected
  - A2C: Advantage actor critic
    - computes Q values (action probabilities) softmax
    - critic network has no actions, one output neuron R - State Utility
      - ↳ does value iteration / back prop.

$$r + \gamma U(s') - U(s)$$

- A3C: Asynchronous Advantage Actor Critic