Big Picture Recap:

- Supervised learning
  - learning from training set of labeled examples
  - try to generalize from seen data to situations not seen in training
  - not necessarily well suited for interactive problems
- Unsupervised learning
  - Finding hidden structure in unlabeled data
  - Still not really the same as interactive where we eventually have a result

Reinforcement Learning

- learn what to do to maximize reward
  - ↳ how to interact with environment
  - ↳ what actions to take
- Characteristics
  - trial and error
    - ↳ not told which actions to take
    - ↳ must discover reward by trying
  - closed loop → actions influence later decisions
  - deferred reward
    - ↳ rewards play out over extended time
    - ↳ may not see reward until end (win/lose game)

Credit assignment problem

↳ which move actually caused you to lose

Examples:

- AIs for games

  sequence of moves leads to win/loss

  learn to play better by playing

  chess, go, backgammon

- robots

  how to teach a robot to move?

  how to teach an autonomous helicopter to fly?

  UTexas videos

RL Components

- Agent: learning decision maker seeking goal
- Environment: everything outside agent
- state: current status of environment
  - ↳ locations of pieces on game board
  - ↳ position of robot
- actions: possible choices for agent to make
  - ↳ legal movement of pieces in game
  - ↳ direction to move robot
- goal: what is trying to be accomplished

* Want agent to learn best sequence of actions to solve problem (maximum cumulative reward)

- policy: mapping from perceived state to action
  - ↳ how agent behaves at given time
  - ↳ determined via
    - lookup table
    - search
    - computation
  - ↳ may be stochastic
- reward: immediate numerical desirability of state
  - ↳ ex: toddler learning to run falls and feels pain
  - ↳ can be +, −, or 0
  - ↳ must setup rewards to reward overall goal
    - aka, in chess only reward for winning
  - ↳ discount over time
    - reward far in future means less

- Value function: long term desirability of state
  - ↳ expected cumulative reward starting from this state
    - ↳ ex: immediate reward of not hanging out with people now may be negative, but long term value could be higher

* Make and evaluate decisions based on values, but only purpose of values is to get more reward

- Model (optional)
  - ↳ simulation of environment

Reinforcement learning views as MDP

MDP = Markov Decision Process

Comprised of

S = set of all possible states

A = set of actions

P_{sa} = state transition probability distribution

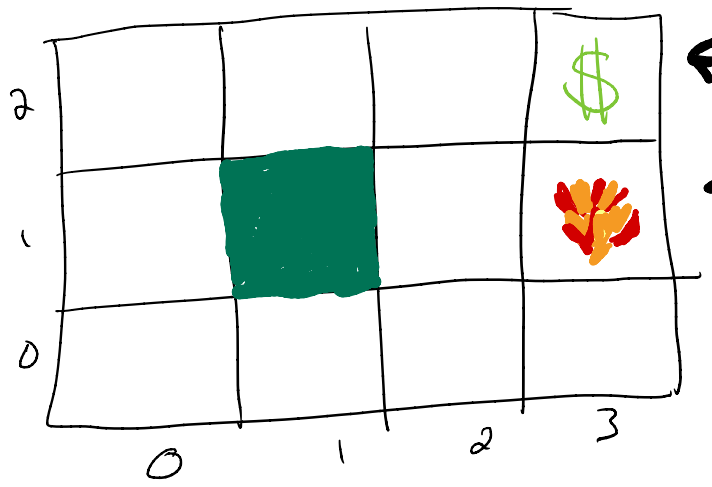
γ = discount factor

R = reward function : $S, A \Rightarrow \mathbb{R}$

- next state and reward depends only on current state and action taken
  - ↳ "Markov property"

Small example: grid world

both $ and fire transition to 0-cost absorbing state → end



← +1 reward

← -1 reward

all other rewards = -0.01

11 possible states

Actions = N, E, S, W

noise = robot doesn't always respond

$P_{21N \to 2,2} = 0.8$ $P_{21N \to 1,2} = 0.1$

$P_{21N \to 3,2} = 0.1$

start at state s_0

choose action a_0

transition to s_1 drawn from $P_{s_0 a_0}$

⋮

Total payoff of sequence $s_0, s_1, s_2, \ldots$

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

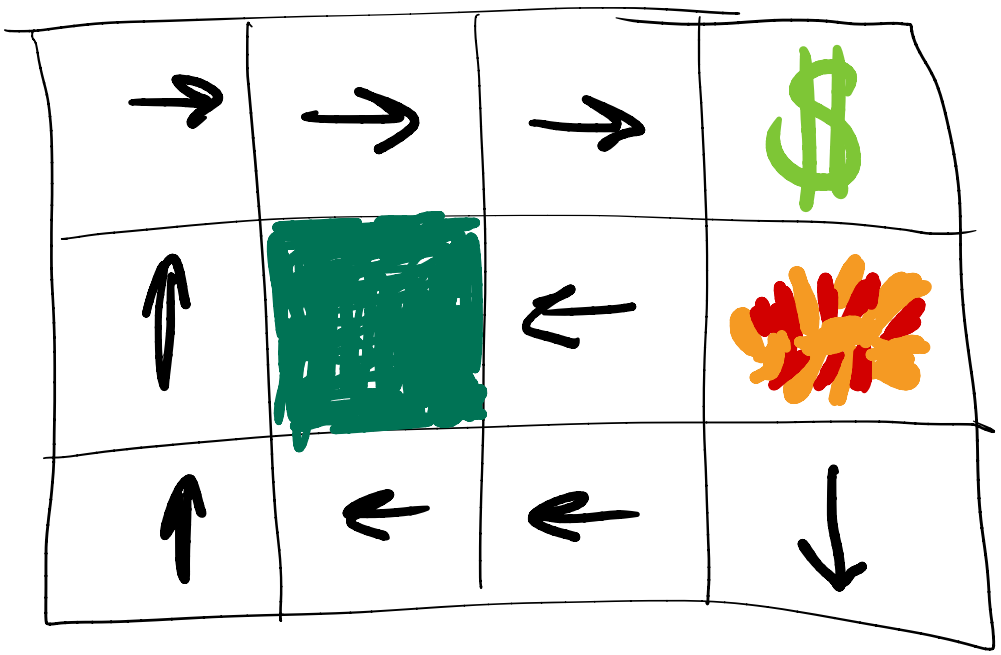
think of gamma like interest rates if dealing with financial (money)

Want to choose actions to maximize

$$E\left[R(s_0) + \gamma R(s_1) + \ldots\right]$$

Policy $\pi : S \to A$

Optimal policy $\pi^* : S \to A$



- changing reward for the states changes optimal policy

  example: making reward more negative encourages taking ~~faster~~ route

More Formally:

Value of State

$$V^{\pi}(s_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

Optimal policy π^* chosen s.t.

$$V^*(s_t) = \max_{\pi} V^{\pi}(s_t) \; \forall \; s_t$$

Can rewrite as Bellman's equation

$$V^{\pi}(s_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, \pi(s)) V^{\pi}(s_{t+1})$$

expected reward of moving to next state

expected cumulative reward from being in next state

Can also work in terms of:

value of state action pairs

↳ how good is it to perform action a_t when in state s_t

↳ written as $Q(s_t, a_t)$

If we know model parameters $p(r_{t+1} | s_t, a_t)$ and $P(s_{t+1} | s_t, a_t)$

- policy iteration:
  repeatedly improve stored policy to get optimal policy
- value iteration:
  iterate to converge to correct V^* values

Assuming we don't know model,
tradeoff between exploration
and exploitation

Exploration: try actions to make better
action selection in the future

Exploitation: choose actions it
has found to be good in the
past to maximize reward

Simple Example: k-armed bandit

Idea: choose from k levers to maximize reward

Explore:

$Q(a) = r_a$ determine by pulling lever

Exploit:

Choose a^* s.t. $Q(a^*) = \max_a Q(a)$

What about nondeterministic rewards?

reward defined by prob. dist. $p(r|a)$

$Q_t(a)$ = estimate of value of action a at time t

↳ average of rewards received for choosing a in the past

$$Q_{t+1}(a) \leftarrow Q_t(a) + \eta\left[r_{t+1}(a) - Q_t(a)\right]$$

delta rule → converges to mean $p(r|a)$

In practice, more complicated than k-bandits

- Q: How to select next action?
  - ↳ tradeoff between exploration and exploitation

Options:

- greedy = choose action with highest value (↱ always exploits)
- ε-greedy
  - with probability ε = choose action uniformly at random
    - ↳ explore
  - with probability $1-\varepsilon$ = choose best action
    - ↳ exploit
  - start with high ε, gradually decrease
- soft-max with decreasing temperature T

$$P(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_i \exp(Q(s,a_i)/T)}$$

small T = favor better actions

TD Learning

Idea: use reward and value of next state to update value for current state

Process

1) Initialize $V(S)$ arbitrarily
2) For each episode
   i) initialize S
   ii) repeat until terminal state
      a) choose action
      b) take action
      c) update $V(S)$

$$V(S_t) \leftarrow V(S_t) + \alpha\left[r_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

$r_{t+1} + \gamma V(S_{t+1})$: better, later prediction

$V(S_t)$: current estimate

$r_{t+1} + \gamma V(S_{t+1}) - V(S_t)$: temporal difference

Q Learning

Idea: constantly update to approximate q_*, optimal action-value function (policy independent)

Process:

1) Initialize table of all $Q(s, a)$
2) For each episode
   i) initialize S
   ii) repeat until terminal state
      a) choose an action
      b) take action
      c) update $Q(s, a)$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$