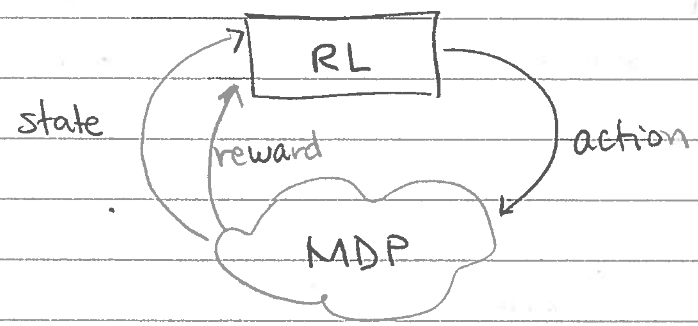
9-1

# Reinforcement learning

What if we have an MDP, but don't know $T(s, a, s')$ or $R(s, a)$?

Instead, we get to interact with it:




Different possible objectives

- Learn optimal policy eventually  ($\pi : S \to A$)
- Maximize summed (discounted) reward over entire lifetime  (hard!)

Two problems

1) How to select action given current learned model?
2) What type of model to learn and how to update it based on experience?

1) Exploration vs. exploitation

Consider a very! simple MDP with 1 state and K possible actions.

- If you take action $i$ you get \$1 with probability $p_i$ and \$0 otherwise

"K-armed bandit"

Tradeoff between trying actions you believe to be highly likely to pay off vs ones you don't know much about.

Interesting theoretical results. We won't study them further.

What to learn?
→ model, policy, or value function?

Experience = set of tuples $(s, a, r, s')$

## Model-based RL

- Estimate $T(s,a,s')$

Simple discrete strategy: $\hat{T}(s,a,s') = \dfrac{\#(s,a,s') + 1}{\#(s,a) + |S|}$

number of times

"Laplace correction"

$$\hat{R}(s,a) = \frac{\left(\sum r \mid s,a\right)}{\#(s,a)}$$

- Now, solve approx MDP or use a finite expectimax to find best action

- Difficult to generalize to continuous (or very large discrete) state spaces. Topic of current research.

## Policy search

Define a functional form $f(s;\theta) = a$ for the policy. Learn parameters $\theta$ from experience.

- $f$ might be a PID controller, where $\theta$ are gains
- $f$ might be a giant neural network, where $\theta$ = Weights

Often $f(s;\theta) = P(a)$

↑ distribution over actions

- Pick $f$ to be differentiable
- Do gradient descent:
  - in low dims, do numeric estimate
    → try $\theta \pm \varepsilon$, run policy multiple times, find $\sum r$
  - in higher dims, there are clever algs (e.g. REINFORCE) but can be hard to make them work reliably.

Good when policy has simple known form.

## Value function learning (most typical)

Remember value-iteration update:

$$Q(s,a) := R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

### Q-learning

"tabular" because we store values in a table (indexed by s, a)

- initialize $Q(s,a)=0 \ \forall s,a$
- $s = s_0$
- loop
  - $a = \text{select\_action}(s)$
  - $r, s' = \text{execute}(a)$
  - $Q(s,a) := (1-\alpha)\, Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a')\right)$

learning rate — $\alpha$; discount factor — $\gamma$

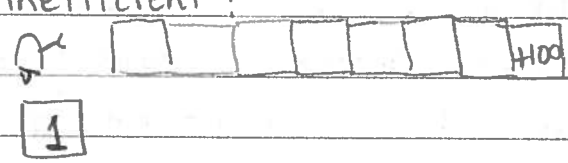
Depends on Experience $(s,a,r,s')$

Can rewrite last line:

$$Q(s,a) := Q(s,a) - \alpha\left(Q(s,a) - \left(r + \gamma \max_{a'} Q(s',a')\right)\right)$$

Looks like a gradient update! (But, actually, it's not. (But it often works to pretend it is.))

Can be very inefficient!




Action selection is important:

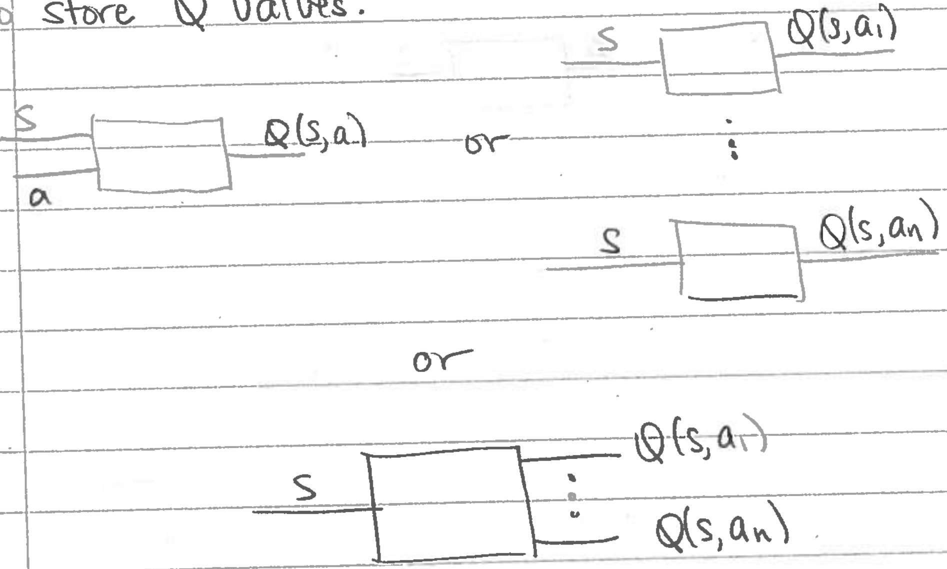
- Some randomness is required or you risk converging on suboptimal policy
- Typical strategy: $\varepsilon$-greedy:
  - with probability $1-\varepsilon$, do $\arg\max_a Q(s,a)$
  - with probability $\varepsilon$, choose a uniformly at random

Guaranteed to converge to optimal Q

- any initialization okay
- any exploration as long as it tries every action infinitely often on an infinite run

9-4 What if $\mathcal{S}$ and/or $\mathcal{A}$ is big?

Use a function approximator (e.g.) neural network to store Q values.




Treat as a regression problem, as if optimizing squared Bellman error as Loss

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

Can be unstable.

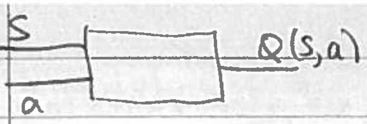
Catastrophic forgetting! (s,a) pairs are temporally correlated (might spend 12 hours in a dark env't and then 12 in a light one, forgetting what to do in the dark).

experience replay: save old $(s,a,r,s')$ experience and mix it in with current data

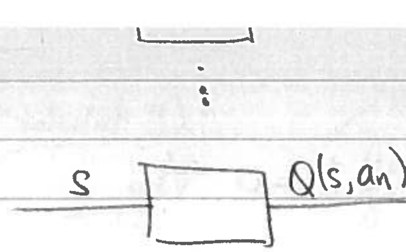
Fitted Q learning

Loop

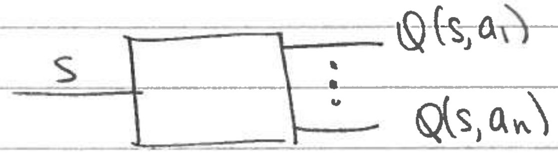
- Use current Q to gather batch of experience
- Add $(s,a,r,s')$ tuples to memory
- For each tuple in memory construct pair
  $\left((s,a),\ r + \gamma \max_{a'} Q(s',a')\right)$
- Train using regular sgd to get new Q




or




or




Treat as a regression problem, as if optimizing squared Bellman error as Loss

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

Can be unstable.

Catastrophic forgetting! (s, a) pairs are temporally correlated (might spend 12 hours in a dark env't and then 12 in a light one, forgetting what to do in the dark).

experience replay : save old (s, a, r, s') experience and mix it in with current data

Fitted Q learning

Loop

- Use current Q to gather batch of experience
- Add (s, a, r, s') tuples to memory
- For each tuple in memory construct pair
  $\left((s,a),\ r + \gamma \max_{a'} Q(s',a')\right)$
- Train using regular sgd to get new Q

test_easy_grid_learn

test_small_grid_learn (update_rate, num_steps)
test_grid_walls_learn
0, 100 clim

9-5

## Atari games

Input (s) : 4 downsampled screen images (60 hz)

Output: discrete joystick command

Q learning + experience replay + NN

Surprisingly effective

- not at long-term planning $\gamma = 0.99$
- not good when partially observable

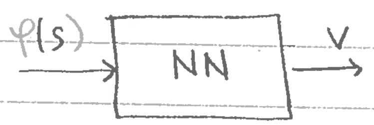
## Alpha zero

Two-player zero-sum complete information games can be solved using value iteration (minor variant).

Takes advantage of the fact that $V(s)$ for player 1 is basically $-V(s)$ for player 2.

Q-learning works, too

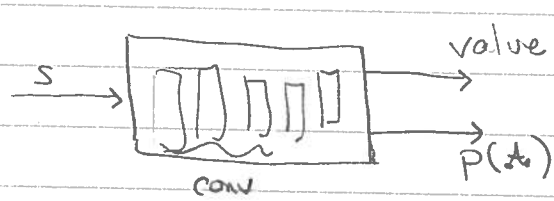
1992: TD-Gammon learns to play Backgammon at near expert level.




1.5 M games of self play

Uses hand-crafted board features.

2017: Silver et al use similar ideas at much larger scale.




Use tree search to improve value & generate supervised training data.

Learns to play Go, Chess, Shogi as well or better than best computer players.