WEEK 3: Reinforcement Learning

<u>Example</u>: <u>Autonomous helicopter</u>
Task: 10 times per second, decide how to move the control <u>sticks</u>.
How: Map $\boxed{\text{State}}$ <u>s</u> $\rightarrow$ $\boxed{\text{action}}$ <u>a</u>

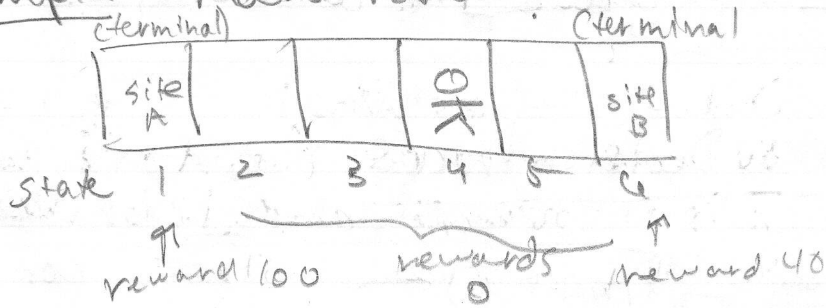
You could use supervised learning to learn the mapping $s \rightarrow a$. But it turns out the "correct" action is ambiguous.

<u>Key input</u>: $\boxed{\text{Reward function}}$ (like training a dog)

e.g. +1 for every second flying well,
-1000 for badly

<u>Applications</u>:
- controlling robots
- factory optimization
- playing games
- stock selling (e.g. to avoid crashing the price)

<u>Example</u>: Mars rover!



at each step rover can go left or right
Go left: $(4, \leftarrow, 0, 3) = \underline{(s, a, R(s), s')}$

How to know if a set of rewards is better than another? $\boxed{\text{Return}}$

Return for actions producing rewards $R_1, R_2, \ldots, R_n$ with $\boxed{\text{discount factor}}$ γ :

$(\gamma < 1)$

$$\boxed{R = \sum_{i=1}^{n} \gamma^{i-1} R_i}$$

This prioritizes actions that give high rewards sooner (i.e. discount later rewards)

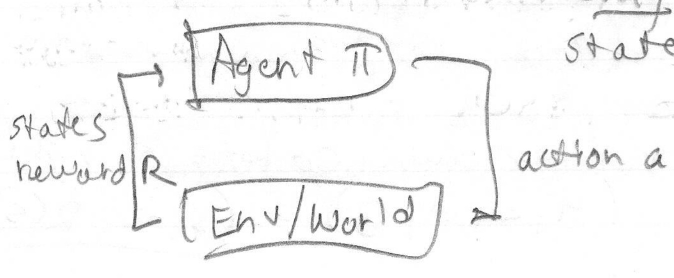
γ maps well to interest rate: time value of money

Note that negative rewards would cause the algorithm to delay the rewards!

$\boxed{\text{Formalization}}$

Goal: define a $\boxed{\text{policy}}$ $\pi : s \to a$ to maximize the return.

Markov Decision Process: future depends only on the current state, not history!



2.5+

State-action value function

Key quantity computed by algs. Learn

$Q : (s, a) \to \boxed{\text{return}}$ ie taking action a once w/state s and the behaving optimally after

"Q function"
(sometimes Q^*)

{Seems circular!}

- Can compute this and then pick actions. (Recall there are multiple acceptable end states, so just pursuing the best outcome isn't necessarily the best return.)

- Let $\pi(s) = a$ where $Q(s,a) = \max_{a'} Q(s,a')$

Bellman Equation

Let s = cur. state, $R(s)$ = reward of cur. state
a = action taken in s
s' = state after taking a
a' = action taken in s'

Then $\boxed{Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')}$

(obvious)

Continuous State Spaces

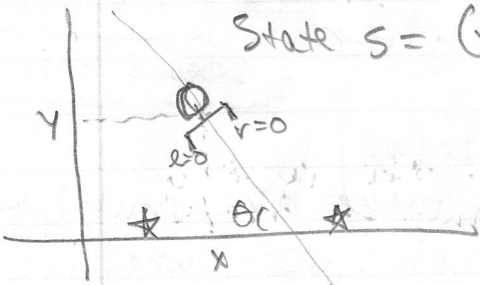
E.g. helicopter, state $= (x, y, z, \underset{\text{roll}}{\phi}, \underset{\text{pitch}}{\theta}, \underset{\text{yaw}}{\omega}, \underbrace{\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\omega}}_{\text{velocity}})$

where $x, y, z, \phi, \theta, \omega : t \to \mathbb{R}$

Or, Lunar Lander:

Actions = { do nothing, fire left thruster, fire right thruster, fire main }

State $s = (x, y, \dot{x}, \dot{y}, \underbrace{\theta, \dot{\theta}}_{\text{angle}}, \underbrace{l, r}_{\text{whether left/right legs are on the ground, } \in \{0,1\}})$



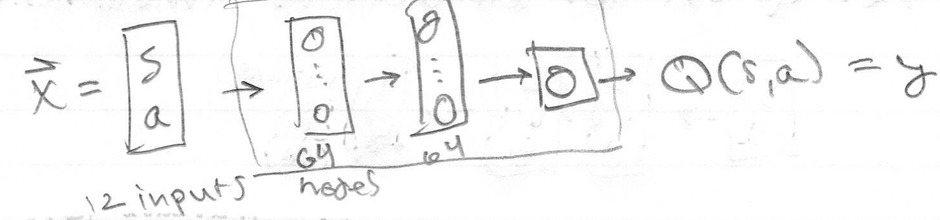
Reward fn:	
at landing pad	+100 – 140
toward/away from	addl.
crash	−100
soft land	+100
leg grounded	+10
fire main	−0.3
fire left/right	−0.03

(but choice of reward fn determines what is learned — incentives + costs)

$\gamma = 0.985$

Learn policy π s.t. given s, $a = \pi(s)$ maximizes return.

How? Train NN to approximate Q. $\boxed{\text{Deep RL}}$



where s has 8 dims. and a is a one-hot encoding of $a \in \{\text{nothing}, L, R, \text{main}\}$.

→ Given state s, compute $Q(s,a)\ \forall a \in A$, and pick the a that maximizes.

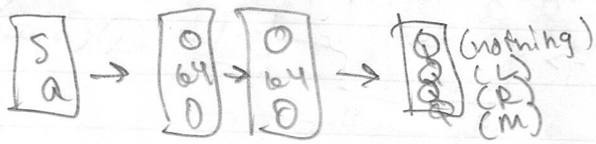
Now use supervised learning to train the NN. But how to get training data?

→ Take a bunch of random actions in order to observe samples $(s, a, R(s), s')$. (Here, fire thrusters and observe outcomes.) And take <u>Q random</u>.

1. Init NN randomly as guess of Q.
2. Loop:
   a. take action in lander, get $(s, a, R(s), s')$
   b. store 10k most recent samples (replay buffer)
   c. (occasionally) train NN for Q_{new}
      $\vec{x} = (s,a) \quad y = R(s) + \gamma \max_{a'} Q(s', a')$
      where Q is the previously trained Q
   d. set $Q = Q_{new}$.

(Deep Q Network)

Improved architecture: output $Q(s,a)\ \forall a \in \mathcal{A}$, i.e. Q val for all actions simultaneously.



Further refinement: $\boxed{\varepsilon\text{-greedy policy}}$.

Recall step 2a: "take actions."

But what? ① Pick a to maximize $Q(s,a)$ (cur. guess)

↳ not best

② Pick $\left\{ \begin{array}{l} a \text{ to max } Q \text{ w/ prob } 0.95 \\ \text{random } a \text{ w/ prob } 0.05 \end{array} \right.$ (ε)

exploitation step

exploration step

↳ better!

Why? <u>Reduce impact of pathological random init. of Q!</u>

Further: start with high ε and lower over time as Q improves.

→ <u>Note</u>: Deep RL can be more sensitive to hyperparameters.

<u>Current State of RL</u>

- Easier to get it working in sim/game than reality
- Far fewer applications than supervised/unsupervised
- Lots of exciting research directions.