What is Reinforcement Learning (RL)?

- intersection of many different fields of science
- Trying to understand the optimal way to make decisions.
- RL is a subtree under Machine Learning (ML)
- The reward system of the human brain is similar to RL

Why is RL a unique ML algorithm?

- No supervisor, only good or bad actions.
- Delayed feedback
- The agent's actions affect the data it receives
  - Think of a car trying to go up a hill. If the car moves farther away from the hill, it receives a different distance than if it is closer to the hill. The car is the agent, & the distance is the data

What is RL used for

- Playing games (like Mario or Frogger)
- Training helicopters how to fly
- Becoming better than humans
  - RL isn't ever taught how to figure things out. It takes around 3-4 days of compute time to become better than a human at games.

How does RL work?

- Rewards. The agent (the thing being trained) gets R_t which is a scalar feedback signal.

Examples

- Flying a helicopter: + reward for carrying out manuver
  − reward for crashing
- Playing a game: + reward for winning
  − reward for losing.

We wish to create a 'unifying framework' for all of these different types of problems.

But sometimes we wish to delay rewards

Why?

Because somtimes satifying the instant hurts the long term. Thing of a RL algorithm that invests money. If we had one that prioritized instant gratification, no money would ever be invested because investing means negative rewards.

So we delay the reward so that investing money now means making money later, & therefore + reward

More about Agents:

- There are 'steps' where at each step the agent takes an action, & then sees if it received a + or − reward.
- The enviornment is what the agent obseeves. The agent influences the enviorment by its actions, but we, the programmer, don't change the enviornment.

$$H_t = A_1, O_1, R_1 \ldots A_t, O_t, R_t$$

H is the history
A is the action
O is the observation
R is the rewart

H_t is the total of all the actions, observations & rewards. This is all of the observable variables.

Our goal is to build an algorithm from the History.
So make an algo. that picks an action from the history to help us get more + rewards.

History is unwieldy, & takes too long. So we wish to replace this history by some concise summary that detemines what happens next.

This concise summary is called a State.

$$S_t = f(H_t)$$

↳ State is any function of the history

The agent doesn't see the whole enviornment. It only sees what is around the agent.

More about the state:

An information state contains all useful information about the history. Basically, you don't need the whole history to make an action, only the last few pieces of history.

Ex: Fibonacci Sequence. The Fib. sequence is the last 2 terms summed up. So 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ...

Consider the whole Fib. sequence to be the whole history. We only need 34 & 55 to get the next action, not the whole history.

∴ Once we know the state (34, 55) the rest of the history can be thrown away. We haven't thrown away any useful information.

This is called a Markov State

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, S_2, S_3 \ldots S_t]$$

The enviornment state is a Markov State.

There are two types of Agents.

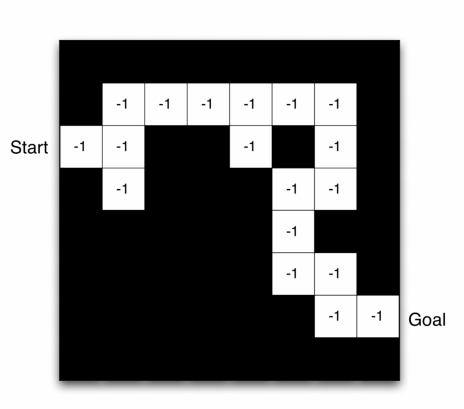
- One that knows the complete enviornment
- One that doesn't, & works from incomplete data
  - ↳ Partial Observability → what we will be dealing with.

What's inside an RL Agent?

- Policy: how the Agent picks its actions
- Value: how good is each state/action
- model: how the Agent interprets the environment

Value → The action determines what type of reward.

Model → Trying to predict the future state & future reward helps the RL improve.



→ numbers represent immediate rewards for a RL algo to solve a maze.

Problems in RL:

- The environment isn't known
  ↳ figures this out through trial & error
- Planning
  ↳ a model of the env. is known
  ↳ agent executes its actions, & then updates its state.

Explore vs Exploit:

It's a balance where mapping out the environment might be long now, but saves time in the future. There is a book called Algorithms to Live By which found the optimal amount of time to explore is 1/3 to 2/3 exploiting

Highly recommend.