Reinforcement Learning

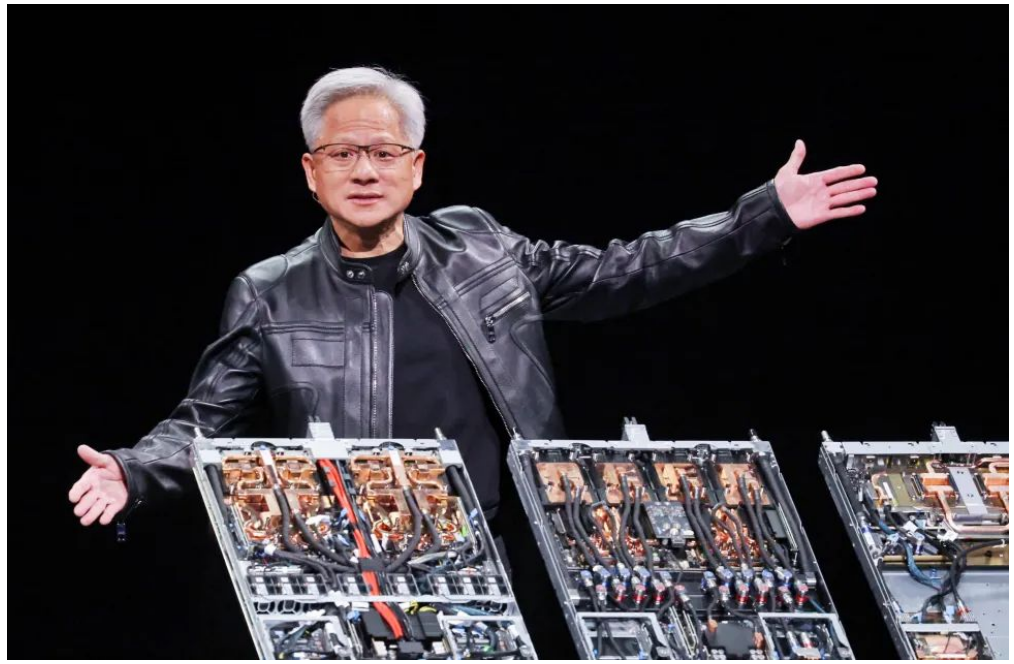
Created By: Bradley Warren
Painstakingly improved by Bhiksha Raj

Outline

- Why Reinforcement Learning? The complex problem
- Timeline
- Markov Decision Processes
- Dynamic Programming
- Model Free Reinforcement Learning
- Function Approximation and Deep RL
- Reinforcement Learning through Human Feedback
- Limitations and Open Problems
- Questions

Why Reinforcement Learning?

- Many real-world problems are sequential decision-making tasks
- Supervised learning assumes static datasets
- Reinforcement Learning (RL): learning via interaction



Why Reinforcement Learning?

Bellman Equation-Principle of Optimality

Policy Optimization

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

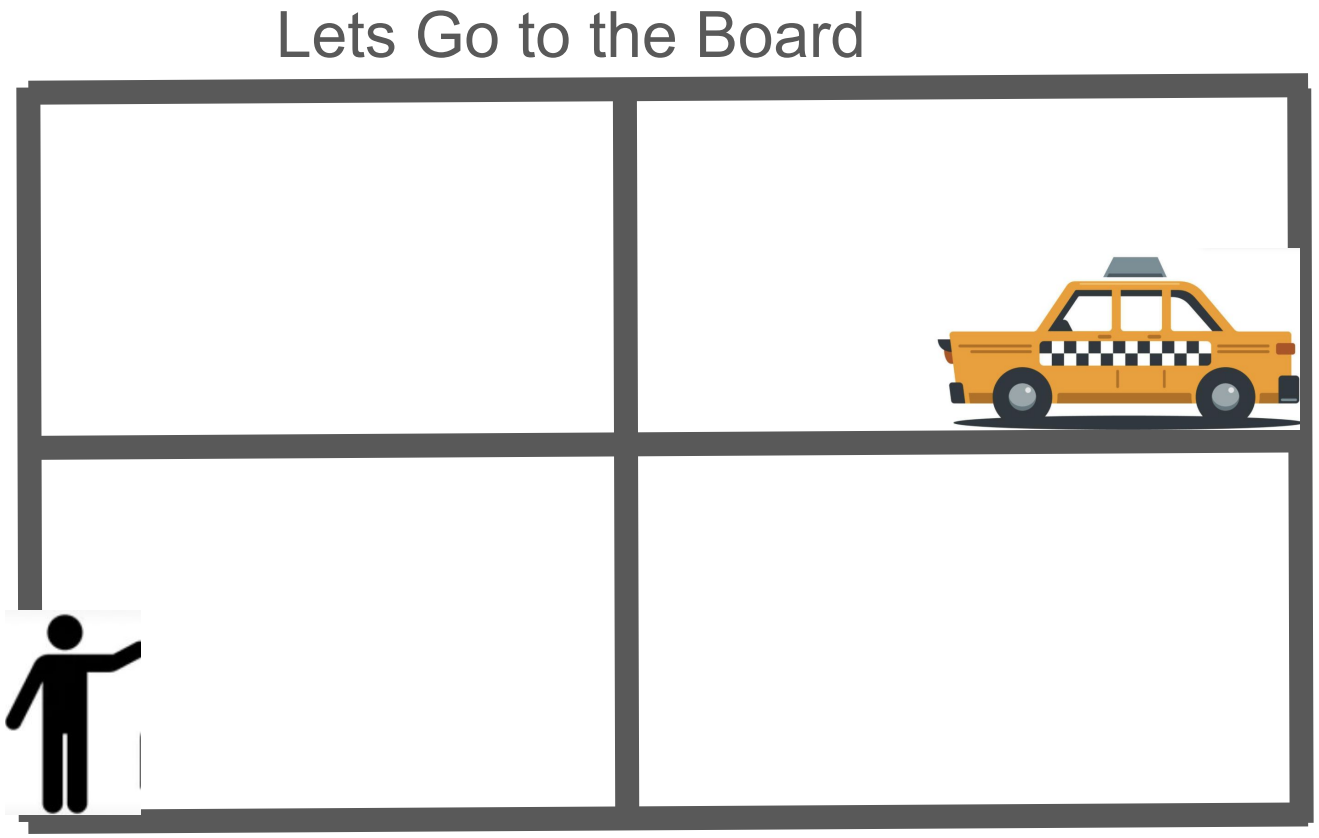
credit Kirk, D. E. (2004). Optimal control theory: An introduction. Courier Corporation.

Timeline

How do I solve the Principle of Optimality?

- Bellman Equation-1949-1957
- Tabular Exact Methods/Dynamic Programming 1970's-1980's
- Monte Carlo Methods 1980's
- Temporal Difference Learning (Sutton)-1988
- Q-Learning (Watkins)-1989
- Function Approximation-1990's-2000's
- Deep Reinforcement Learning-2013-2015
- Policy Gradient Methods- 2000 resurgence 2015
- Actor Critic Methods-1983(Sutton)-Deepmind 2016
- Trust Region and Stable Policy methods 2015-2017
- Model Based RL Resurgence 2018
- RLHF-2018

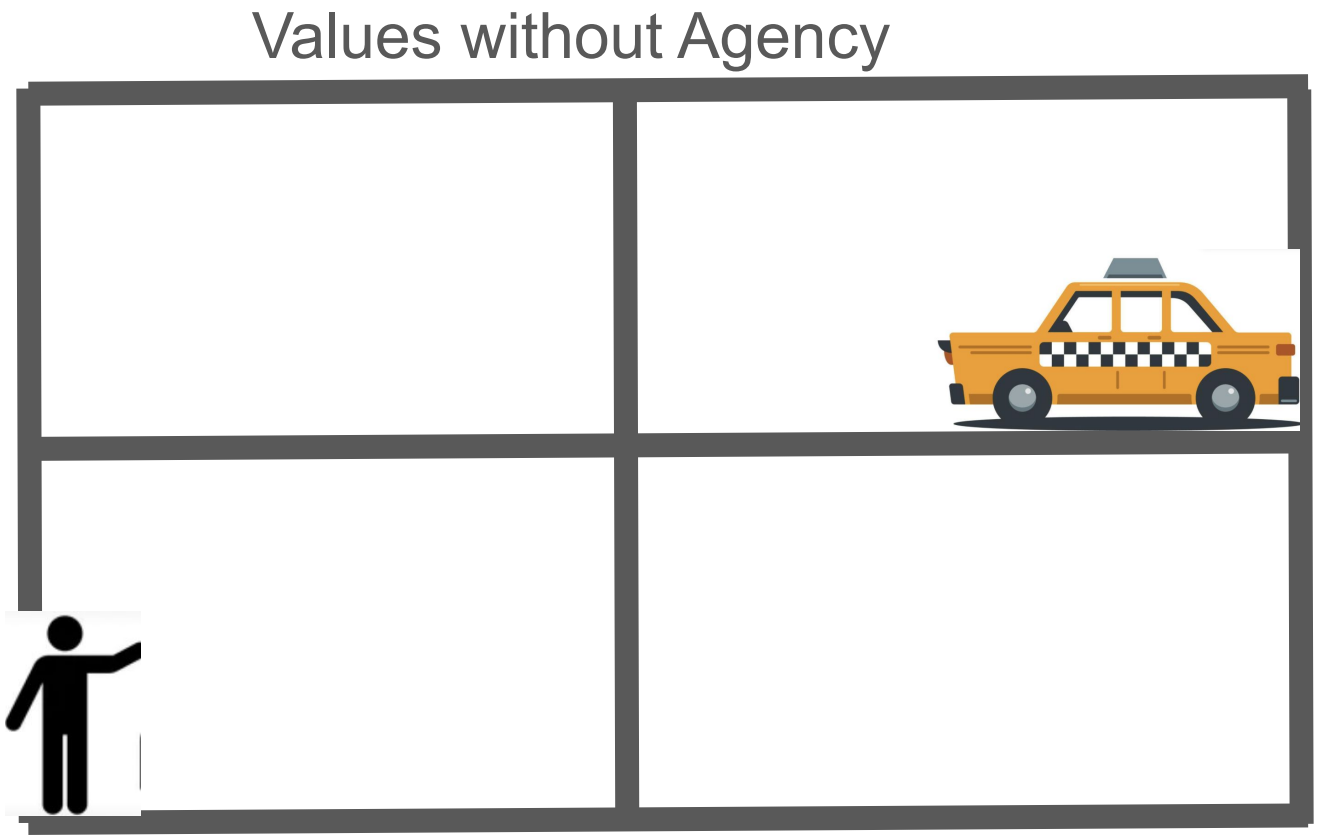
Introducing our problem



The Story So Far

- State
- Reward
- Cost
- Objective

Introducing our World



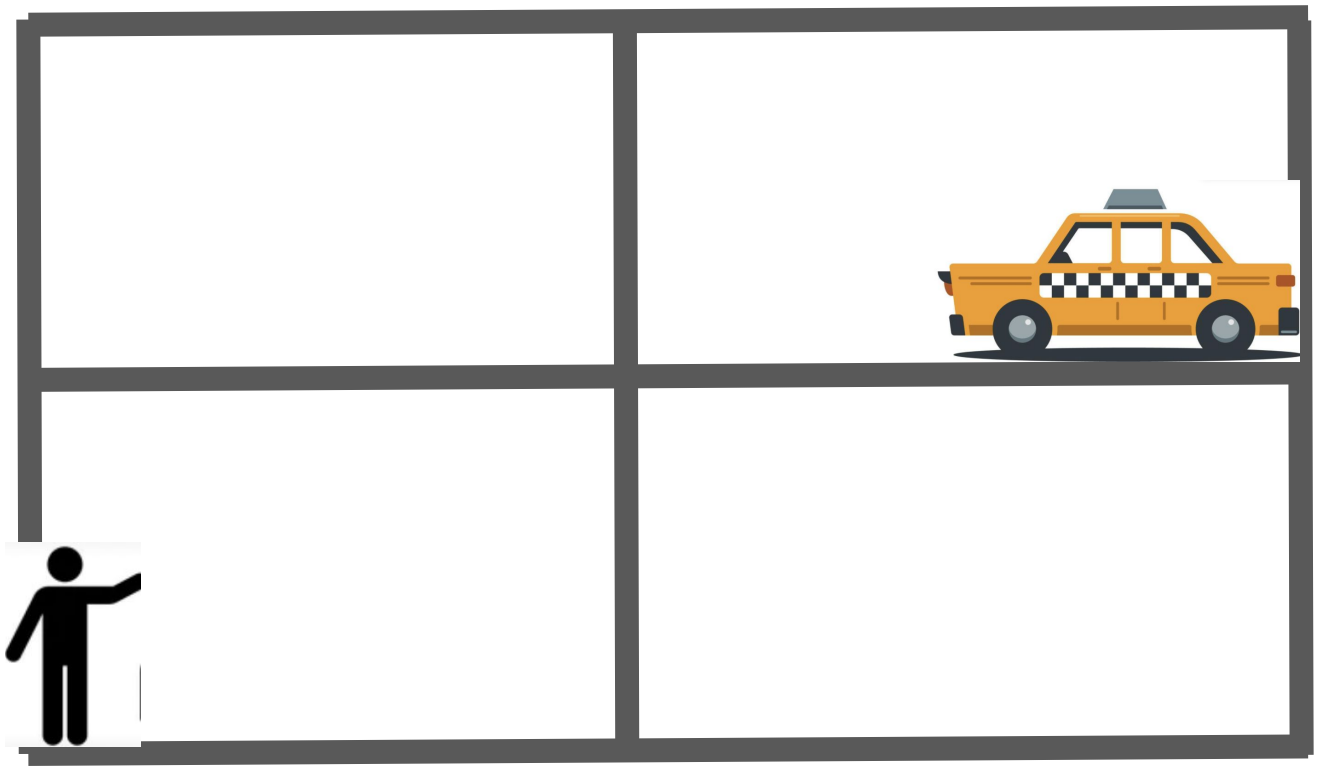
The Story So Far

- No Agency model
- Bellman Equation
- Value Function
- Markov

$$V(s) = \max_a \sum_{s'} P(s'|s,a)\,[R(s,a) + \gamma V(s')]$$

Markov Process: Information State

Information State Can Be different than physical state



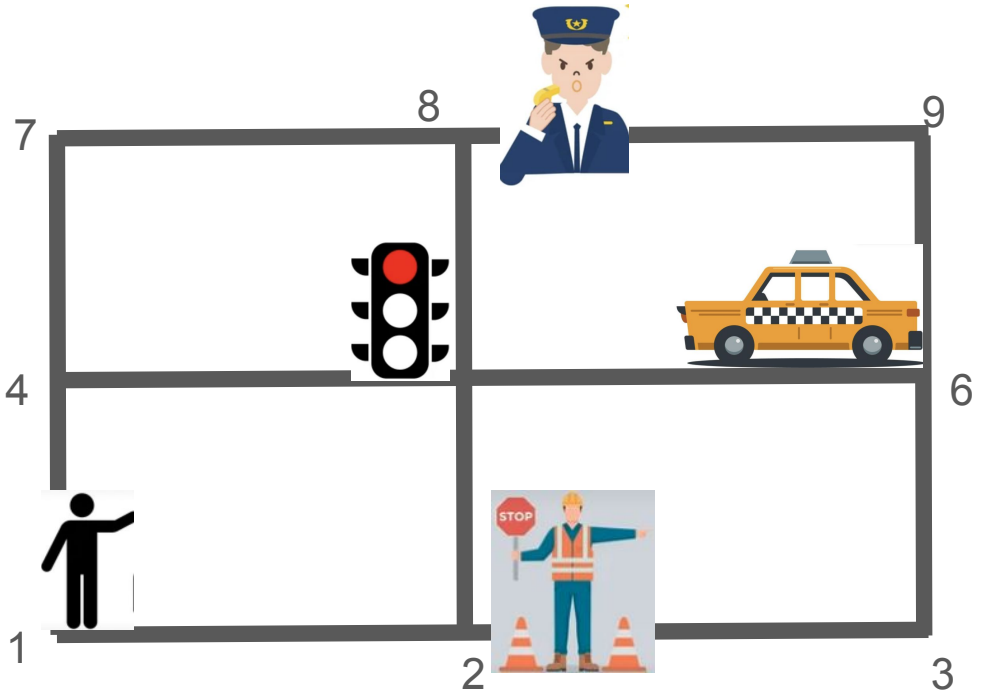
The Story So Far

- Information State
- Reframing a state
- Markov to model the world

Introducing Agency

Movement is no longer random

We must learn where to move

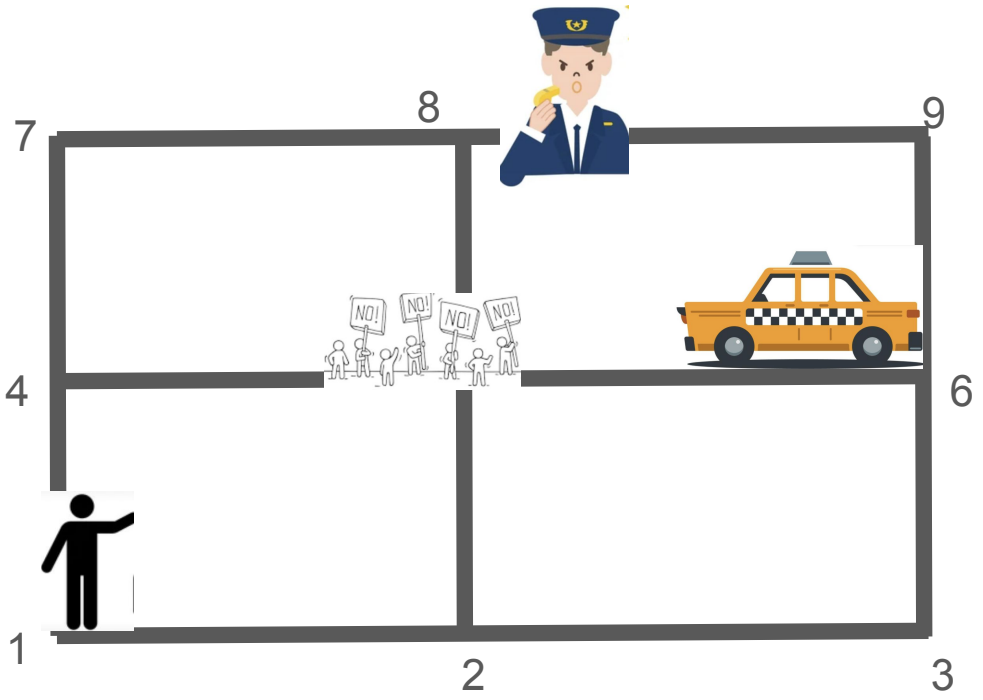


The Story So Far

- Policies
- Decisions

Markov Decision Process

Environment Changes in response to action



The Story So Far

- Markov Decision Process
- Modeling a Changing Environment
- Deterministic
- Stochastic
- Discount

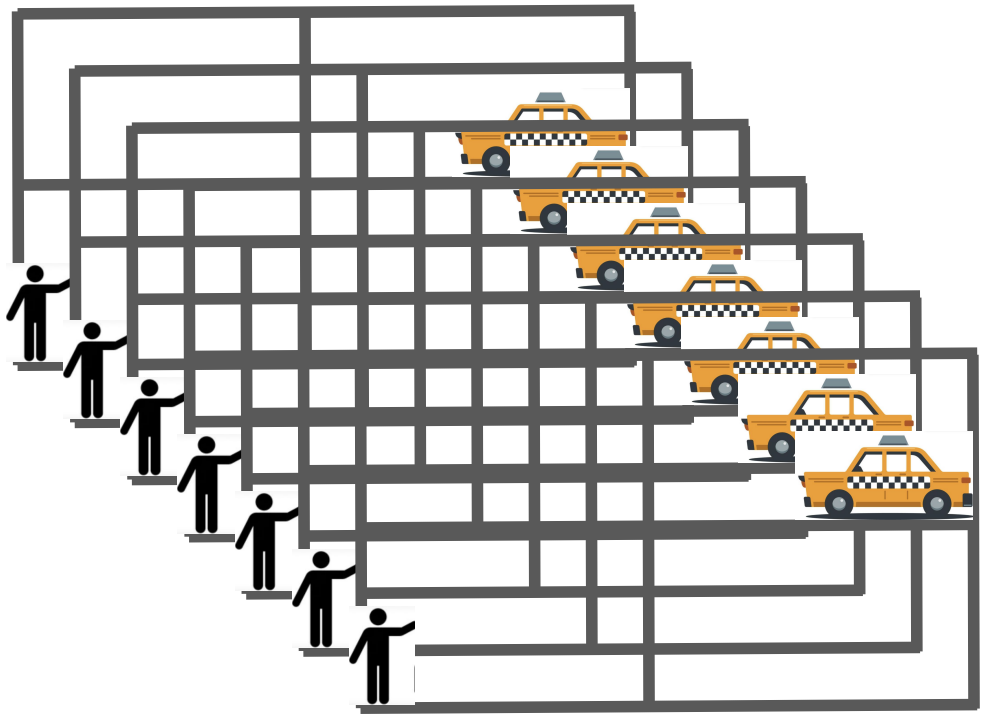
Computing the MDP

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- Given the expected rewards at every state, the transition probability matrix, the discount factor and the policy:

$$\mathcal{V}_\pi = \left(\mathrm{I} - \gamma \mathcal{P}_\pi\right)^{-1} \mathcal{R}_\pi$$

- Matrix inversion $O(N^3)$; intractable for large state spaces

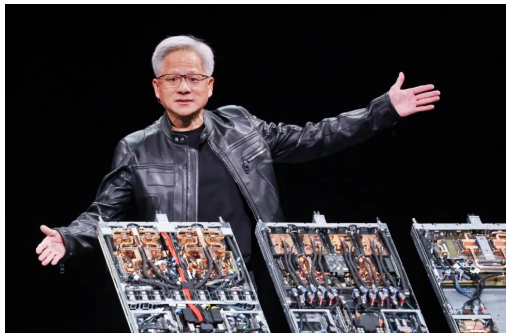


Looking Back

S: state space

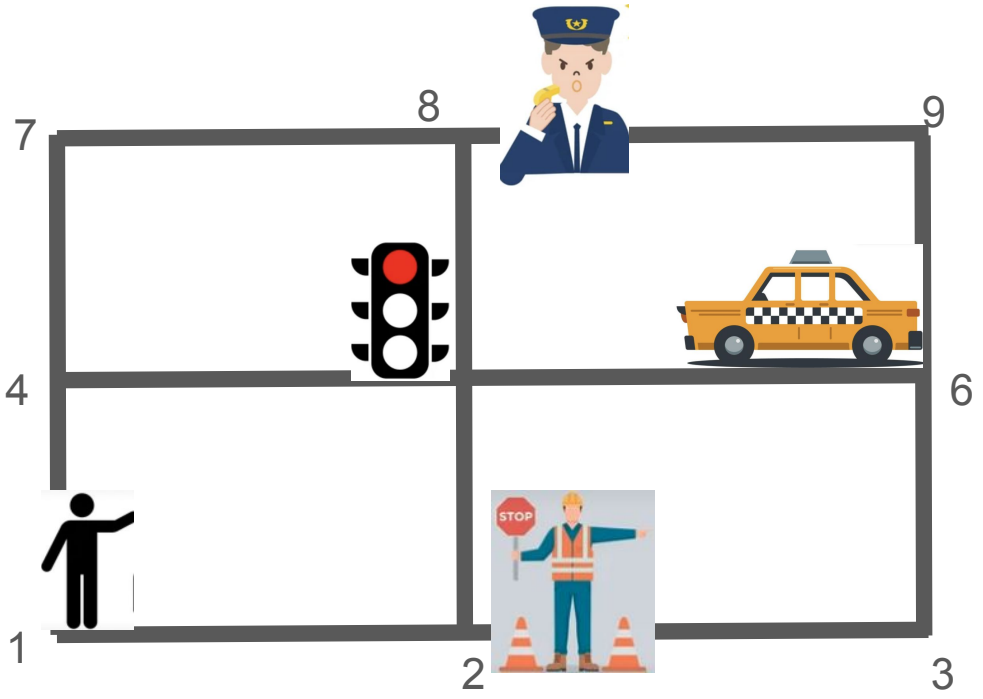
A: action space

s′ Best State



Model Free RL

Dealing with a changing or unknown environment



What is Model Free RL

No Transition Model

No Environmental Knowledge

Learn From

- Interaction
- Samples

Explicit Trial and Error

Policy Evaluation and Policy improvement

Monte Carlo Simulation

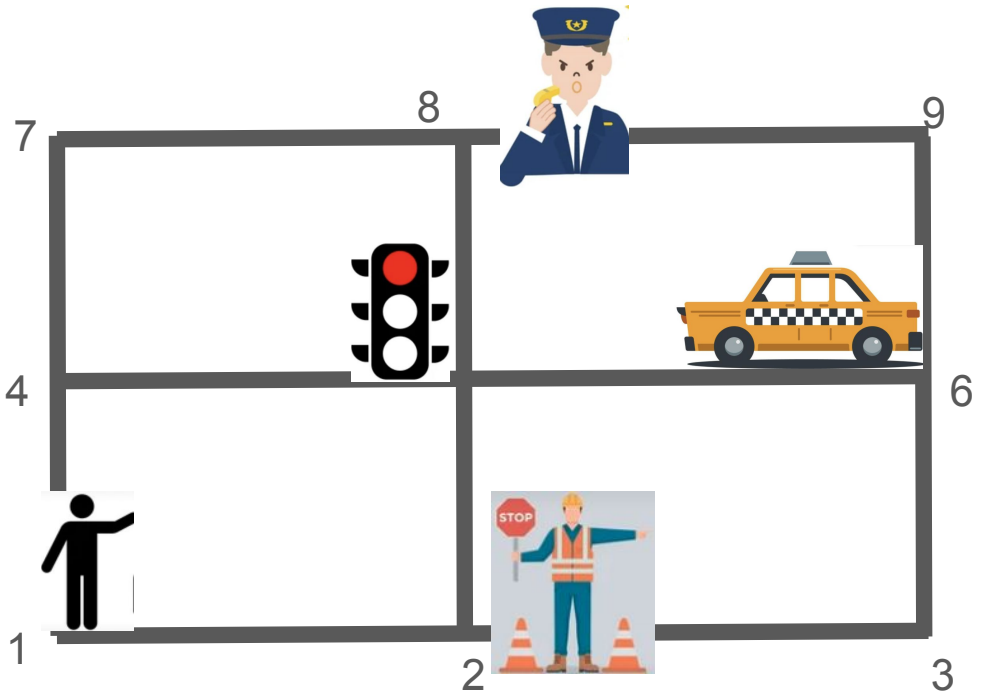
Learn from full outcomes

- Run through to the end
- Observe Total Return
- Update Value Estimate

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

Monte Carlo Simulation

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$



Monte Carlo Simulation

New Estimate = old Estimate + Small Correction

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

So Simple I love it!!! We're going to be rich!!!

Why Not Monte Carlo?

- Good:
  a. Will eventually get to the right answer
  b. *Unbiased* estimate
- Bad:
  a. Cannot update anything until the end of an episode
     i. Which may last for ever
  b. High variance! Each return adds many random values
  c. Slow to converge

Temporal Difference Learning

Learn as you go

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

- Update immediately
- Uses:
  - Reward + next estimate
- This is bootstrapping

Why not Temporal Difference Learning?

Learning from a guess

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

What if s_{t+1} is wrong?

Monte Carlo vs Temporal Difference

Wait until we know the truth?

Estimate the truth from my current understanding?

- Cascading Error
- Can Diverge
- Can Oscillate
- No Ground Truth Signal

How Do we fix this?

Moving from Value functions to Action Values

TD learning:

- Learns V(s)

Problem:

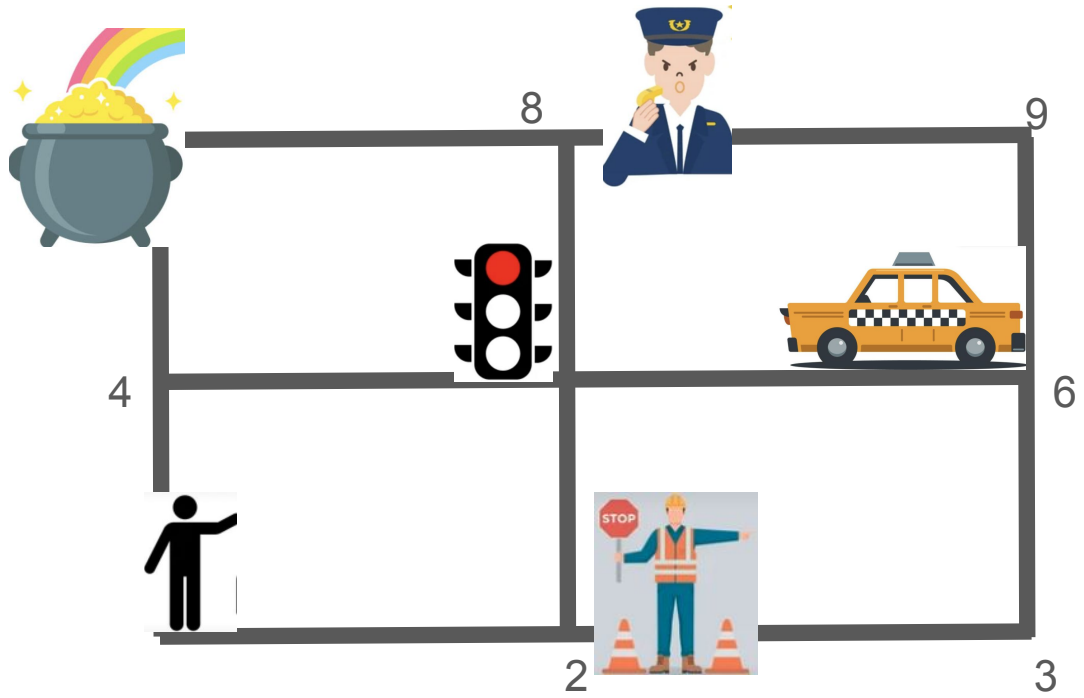
- Still need a policy to act
- Instability with bootstrapping

Let’s learn the value of an action directly

On Policy vs Off Policy Learning

SARSA Vs Q-Learning

Exploitation vs Exploration



SARSA

Pick a random initial policy .

Repeatedly create episodes.

For each time step in the current episode:

Start at state	(S)
Carry out action	(A)
Get reward	(R)
Reach state	(S)
Estimate optimal future action	(A)

Estimate optimal future return

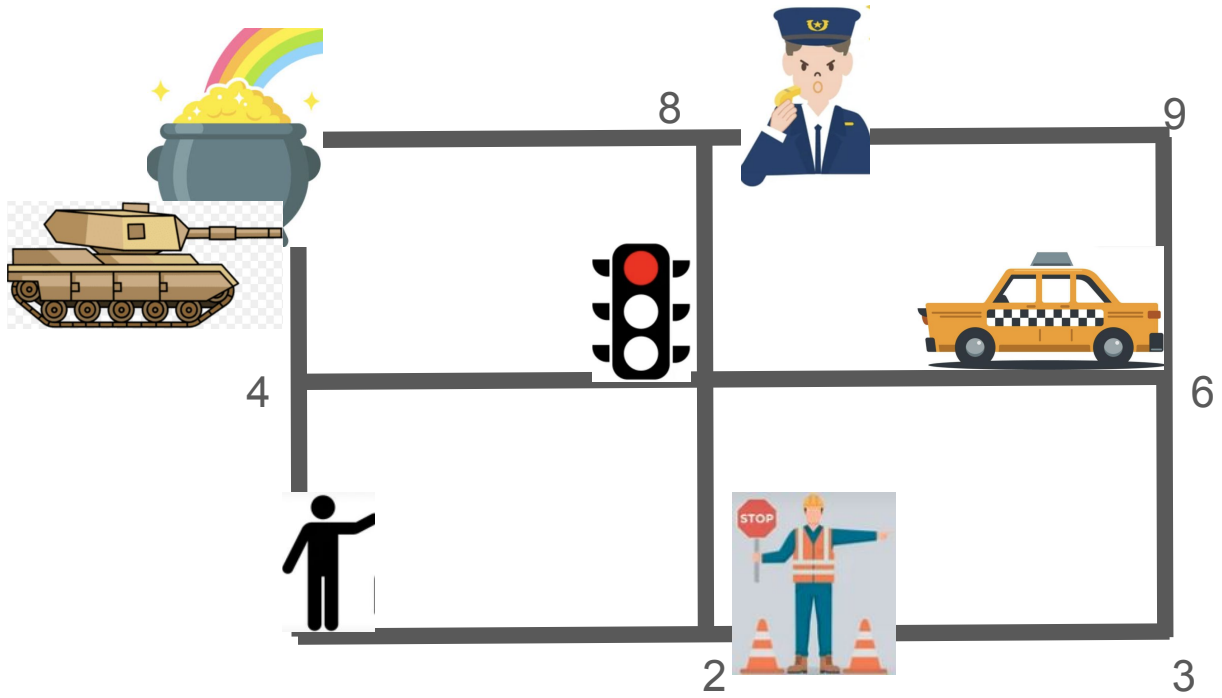
Update Q(S,a)

Update the current policy

SARSA

Never Learning from something that isn't already in a policy

Need to consider factors around a policy



Q Learning

Q(s,a):

Learn the value of taking action a while in state s

- No model required
- Directly learns optimal behavior

Learning What to do, not just how good things are

Q Learning

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

Current guess: Q(s,a)
Target:

- reward + best future action

Update toward that target

Update towards the best possible future

Q Learning: Why this helps

No need for separate policy
Automatically improves behavior
Off-policy:

- Learn optimal policy while exploring

Off Policy-Learning

Learn from Any behavior

Behavior policy:

- What you actually do (exploration)

Target policy:

- What you want to learn (optimal)

Q-learning learns optimal policy **regardless of behavior**

Perfect…. Or is it?

Still bootstraps

Can:

- Overestimate values
- Be unstable with neural networks

I’m taking Deep Learning it better work with neural networks

Function Approximation and Deep RL

Why previous methods have broken

State Space is too large

Cannot Store Table

- Straight up memorization

Need Generalization

Deep Q Network

Everything We've done so far assumes saving values in a table

What if instead of a table we used a neural network?

Q Learning meets Neural Networks

Tables memorize Neural Networks Generalize

Deep Q Network

Input is the State

Output is the Q values for each action

$$Q(s,a) \rightarrow Q_\theta(s,a)$$

Deep Q Network

Same equation different representation

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

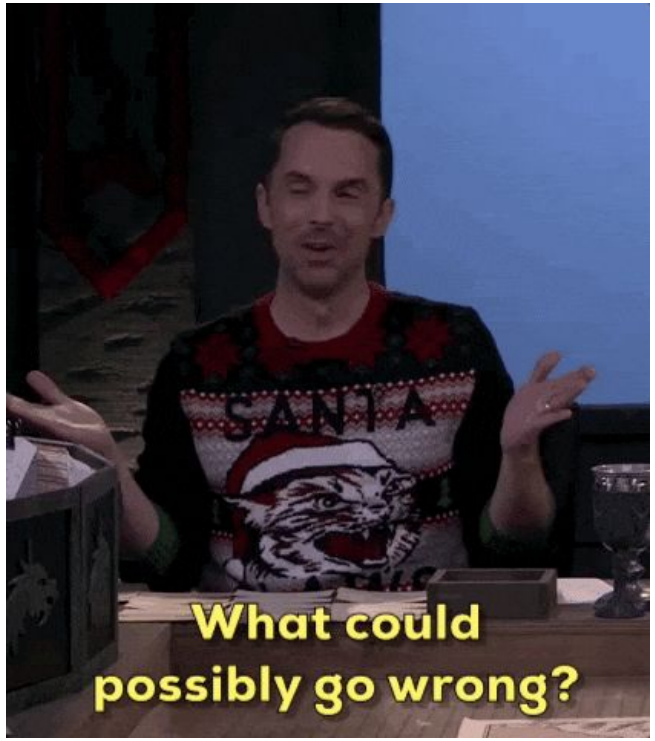
$$Q(s,a) \approx Q_\theta(s,a)$$

We've moved from a table lookup to a function approximation

Ok now we're Perfect?

We took Q Learning replaced the table with a neural network

What could possibly go wrong?



The Core Problem

Feedback system powerful function approximator

- Neural network (nonlinear, global updates)
- TD learning (bootstrapping)
- Streaming data (correlated)

This is an unstable combination

Correlated Data

Sequential Data is highly correlated

- Neural networks assume independent and identically distributed data
- Instead:
  - Similar states
  - Similar updates

Leads to:

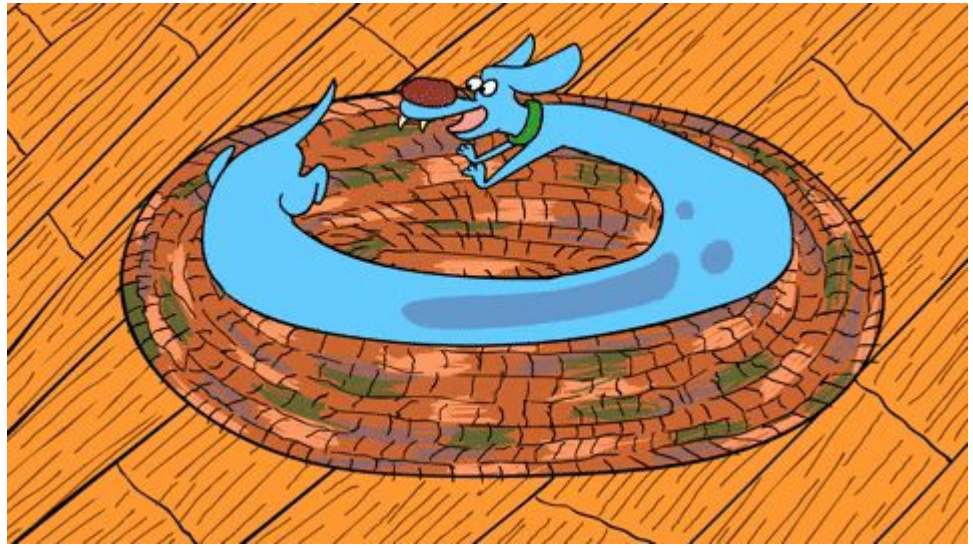
- Overfitting
- Poor generalization
- Instability

Think of this as training on a movie instead of sampled screenshots

Moving Target Problem

We're updating the Same network we're targeting

$$r + \gamma \max_{a'} Q(s', a')$$



Target changes every step

Network chasing itself

Feedback loops

Q-Learning uses its own predictions

Neural Networks Generalize across states

Errors

- Propogate
- Amplify

So We're back to

- Divergence
- Oscilation

So wait we're…

Trying to learn from our own guesses

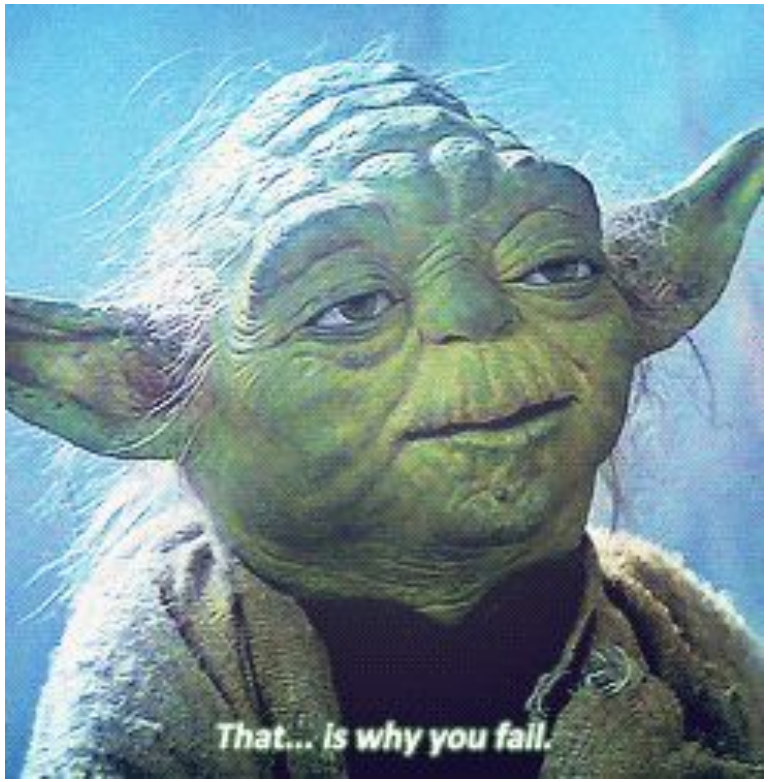
While changing the answer key

Using Highly similar examples

I could have told you that would fail!

Naive DQN

Fails because it learns from unstable targets using correlated data



Is there a solution?

Enter Deepmind (2015)

Genius but simple

Correlated Data: The Fix

Store Past Experiences (s, a, r, s')

Train on Samples instead of Sequentially

We're breaking correlation by mixing past experiences

- Stabilizes training
- Improves data efficiency
- Reuses experience

Moving Target Problem: The Fix

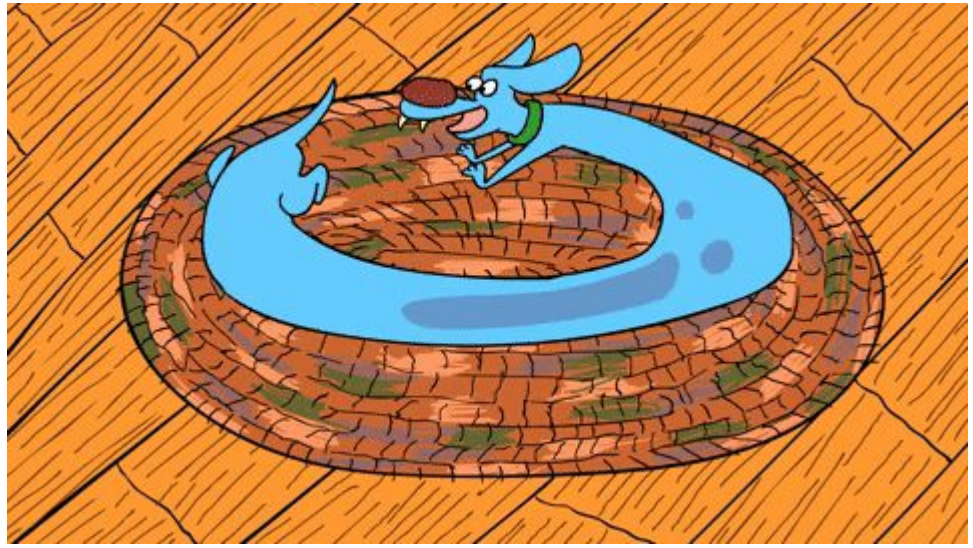
Use a Separate Network

- Copy weights periodically
- Keep target fixed for a while

Slows down learning target

Instead of chasing itself:

- Network learns against a stable reference



Deepmind didn't Change Q Learning

They Made it trainable

Replay stabilizes the data.

Target networks stabilize the objective.

Are We Rich Yet?

Everything has been learning from Environmental Rewards

but…

Real World Tasks don’t have rewards

What if we don’t have a reward function

In Video Games

- Reward=High Score

In Robotics

- Reward=Completed Task

The Real World

- What is the reward for using proper format?
- What is the reward for being helpful?
- What is the reward for a creative task?

Reinforcement Learning

The hardest part of RL isn't Learning
It's defining the reward

So Where do Rewards Come From?

Reinforcement Learning through Human Feedback

Make the Humans do it

Humans:

- Rank outputs
- Give feedback
- Express preferences

This becomes the reward signal

Reinforcement Learning through Human Feedback

Replace Environmental Rewards with Human Preference

- Learn the Reward Model
- Optimize the Policy

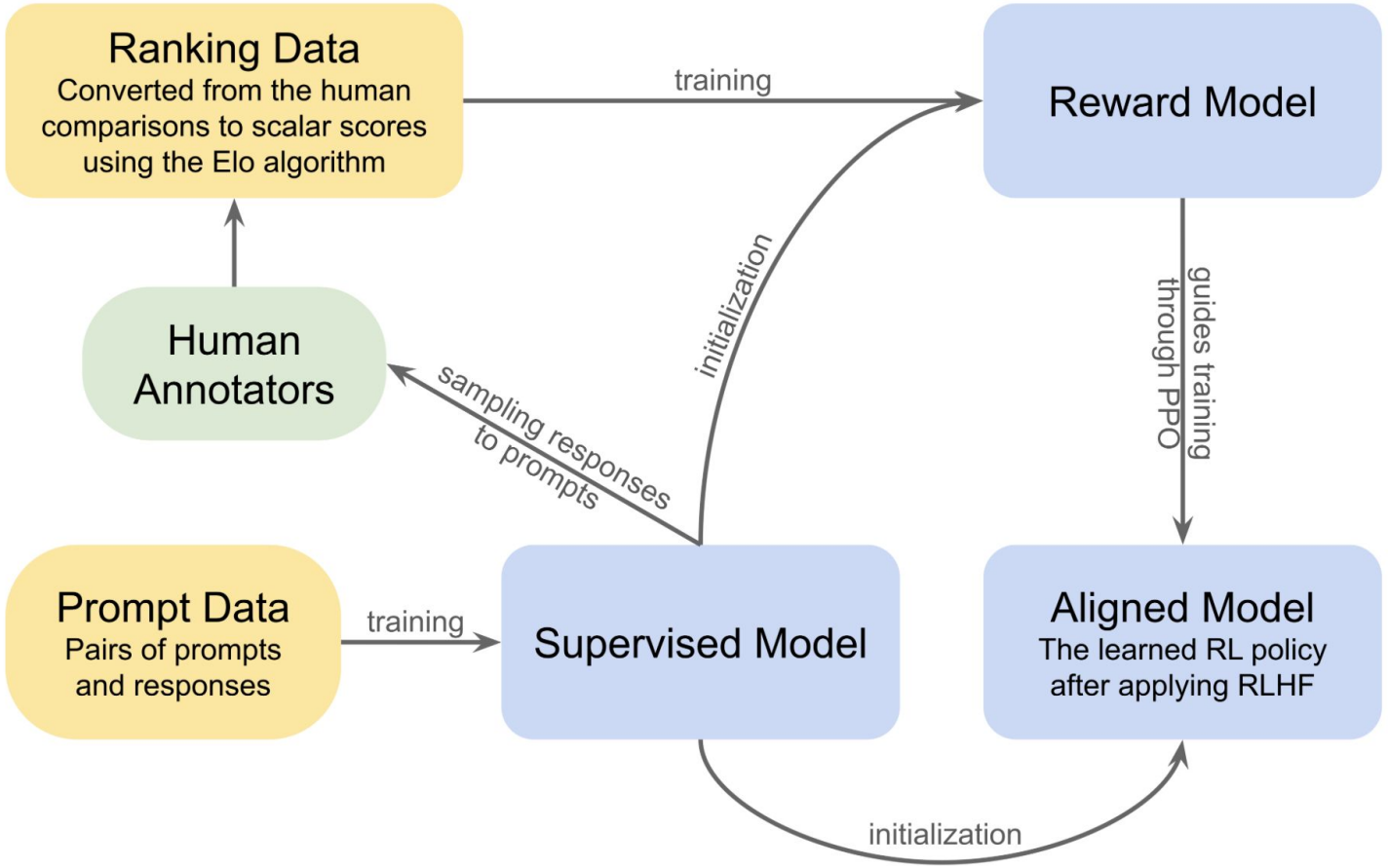
We learn from what humans want

Same Framework, New Reward

- Maximizing Human Preference
  - Learned Objective replaces Fixed Objective

Nothing about RL has changed except where the reward comes from

RLHF Architecture



The Story So Far

Dynamic Programming → known model

Model-Free RL → learn from data

Deep RL → scale with neural networks

RLHF → define rewards using humans

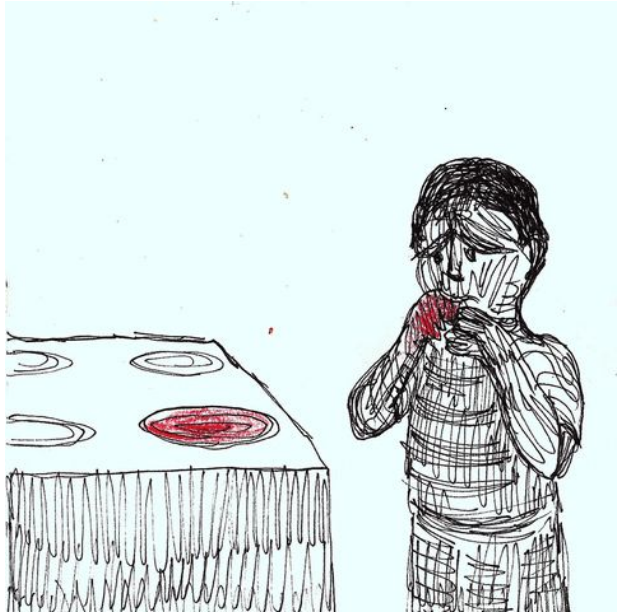
Limitations and Open Problems

RL is very powerful, but far from solved

Problem: Inefficiency

RL needs Millions of Interactions

Humans learn from few examples



Problem: Rewards are hard to define

Story Time

You get what you reward not what you want

US air force denies running simulation in which AI drone 'killed' operator

Denial follows colonel saying drone used 'highly unexpected strategies to achieve its goal' in virtual test

Problem: Exploration vs Exploitation

Try new things vs Doing what you know works

Sparse Reward Environment
Long Horizon Tasks

Problem: Stability and Convergence

Both Neural Networks and RL

Can Diverge
Can Oscilate
- Especially with off policy learning and function approximation
- Training Can be fragile

Problem: Generalization

Simulation vs Real life

Perform well in training
New data can bring in entirely new variables

Problem: Humans Suck

Humans are:

- Inconsistent
- Biased
- Expensive
- LAZY

Story Time

Where is RL Going

- Better exploration methods
- Sample-efficient learning
- Safer RL
- Better alignment (RLHF improvements)
- World models / model-based RL resurgence

Key Takeaways

1. Reinforcement Learning is ideal for multistep tasks

2. Exploitation vs Exploration

3. Rewards and Penalties are key

Quick Recap

RL = sequential decision making

Bellman = recursive value

Model-free = learn from experience

Deep RL = scale with neural networks

RLHF = align with humans

Thank you