# Bootcamp 6: Reinforcement Learning

William H. Guss, James Bartlett
{wguss, james}@ml.berkeley.edu
Machine Learning at Berkeley

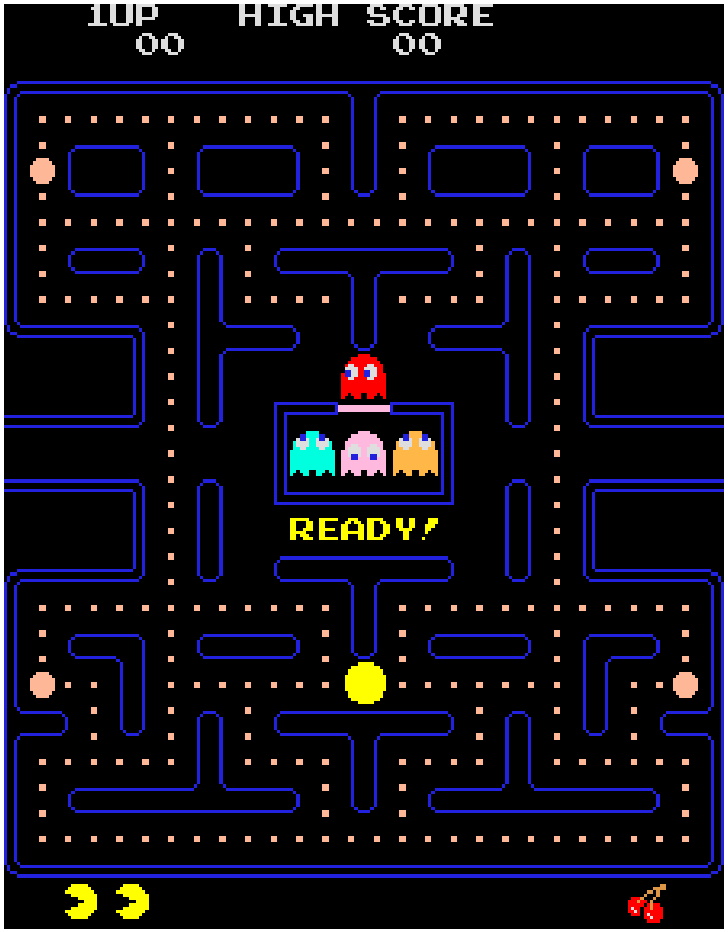
April 22, 2016

# Overview

# Problem: ML for Pacman.



*How would you solve pacman with machine learning?*



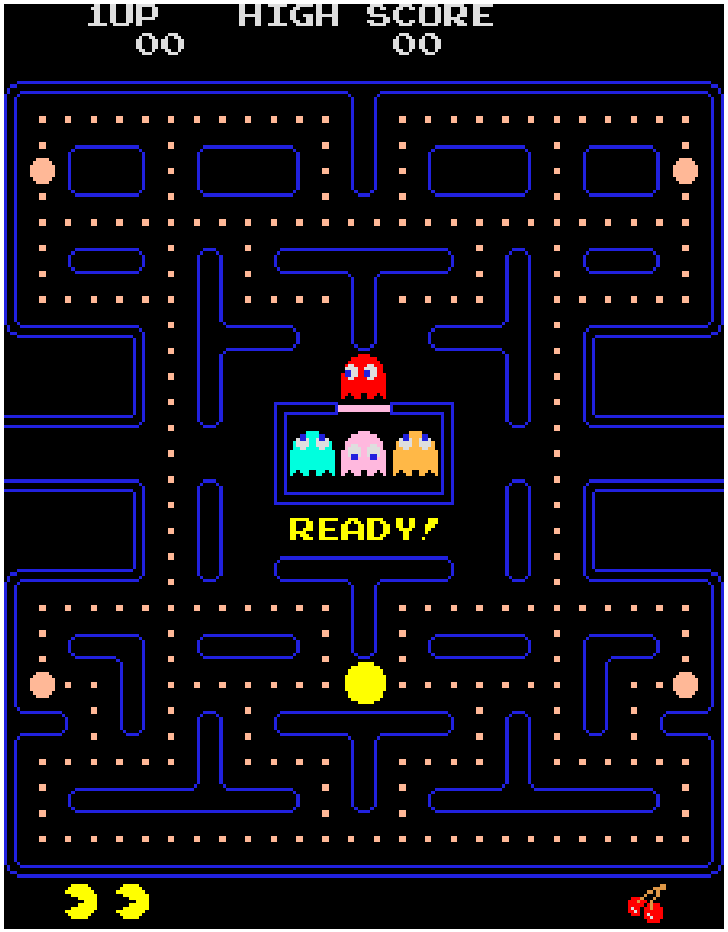
# Problem: ML for Pacman.



*How would you solve pacman with machine learning?*

**Find a model which takes screen pixels to actions:**

$$\pi_\theta : s_t \mapsto a_t.$$



# Problem: ML for Pacman.

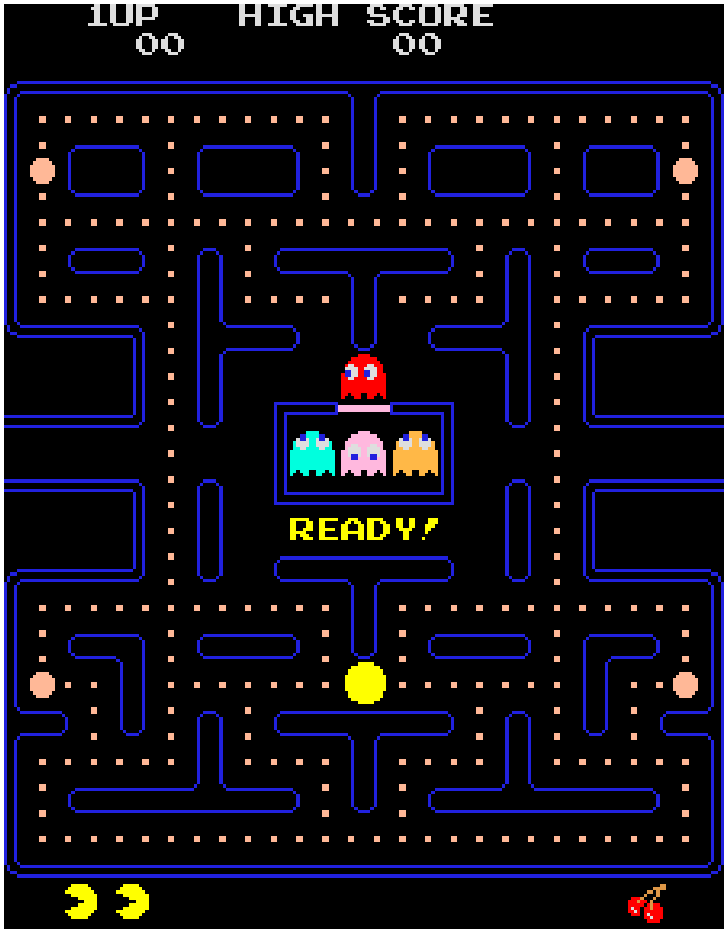


*How would you solve pacman with machine learning?*

**Find a model which takes screen pixels to actions:**

$$\pi_\theta : s_t \mapsto a_t.$$

*What is your loss function? Data?*

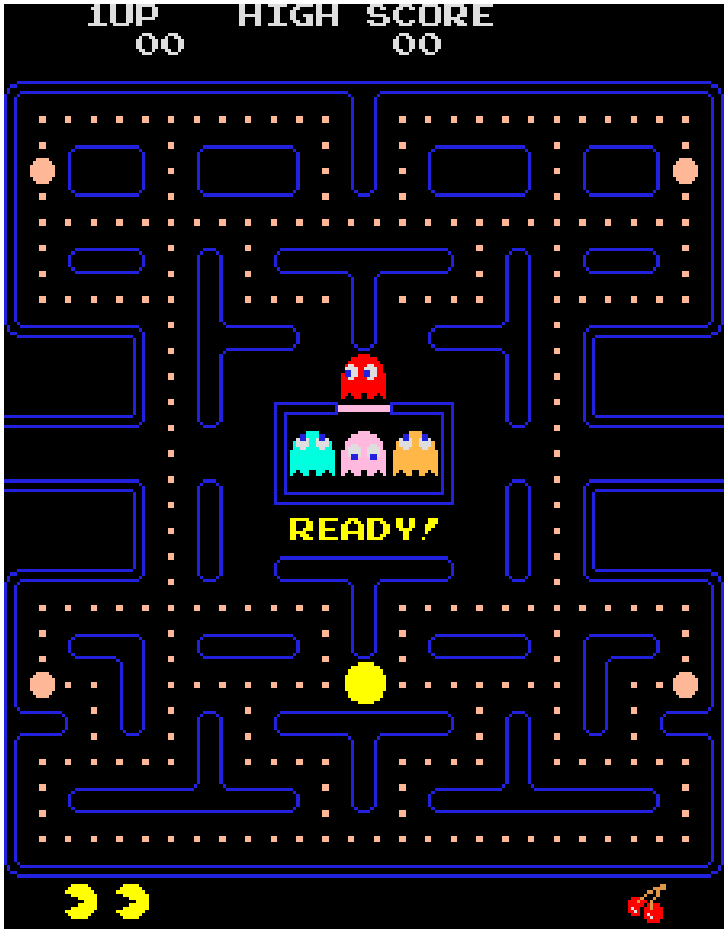


# Problem: ML for Pacman.





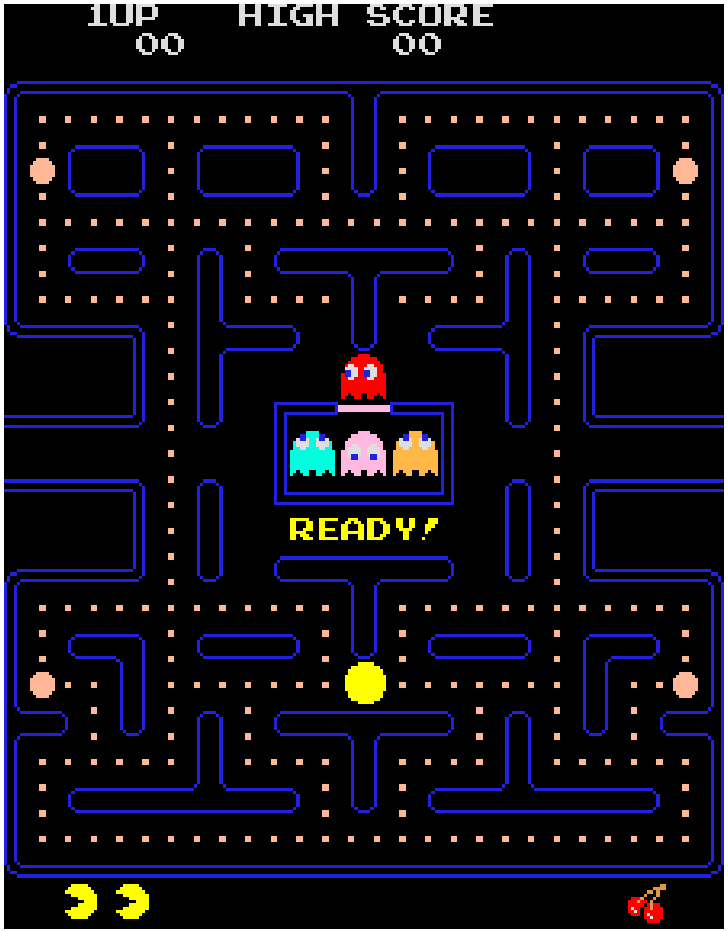
$\Longrightarrow$



# Solution: Reinforcement Learning



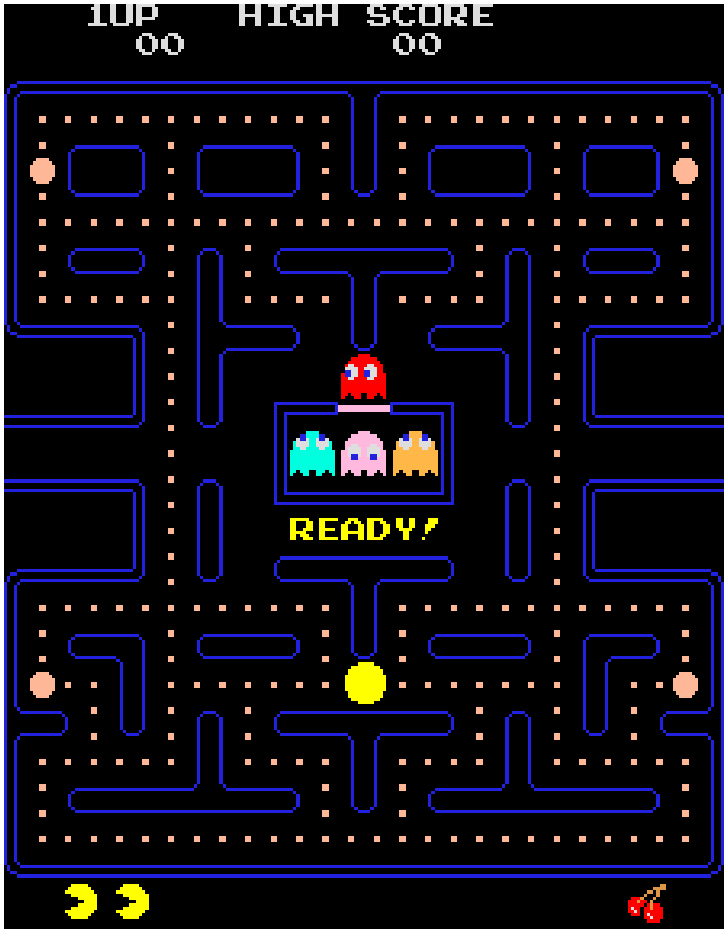
- Supervised learning is *not* the most general formulation of learning.



# Solution: Reinforcement Learning



- Supervised learning is *not* the most general formulation of learning.
- Humans learn through reward and penalty



# Solution: Reinforcement Learning

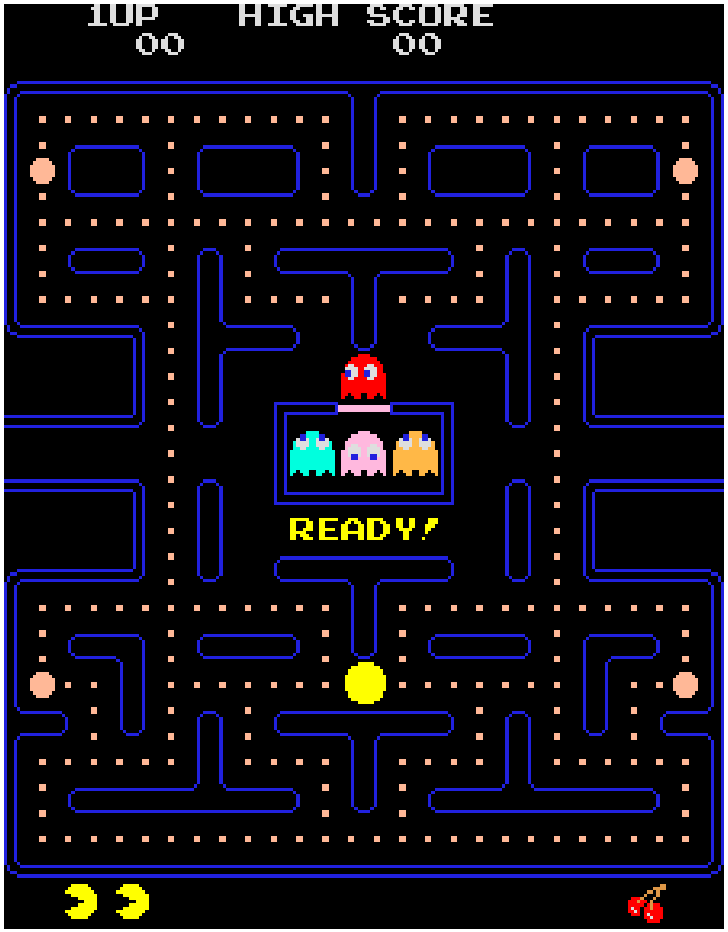


- Can we make algorithms which improve with crude reward signals?

**Machine learning without explicit objective functions**

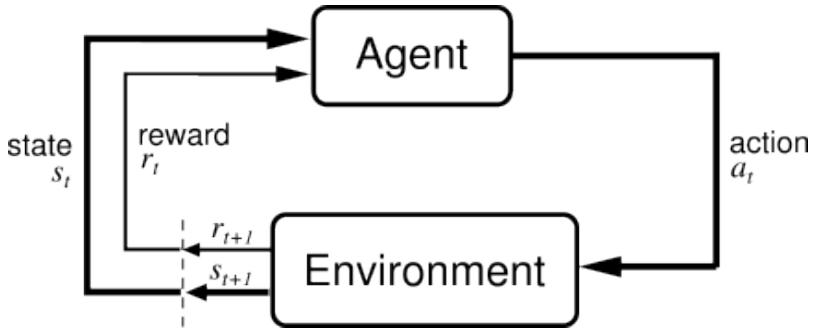
$\Downarrow$

**Reinforcement Learning (RL)**



# The Core Idea






- Models (agents) take action $a_t$ in some environment.
- Environment provides state $s_t$, reward $r_t$.
- Models learn to maximize reward $r_t$, $\forall t$.

# Markov Decision Process (MDP)

ML@B

Environment, $E = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \rho, r)$.

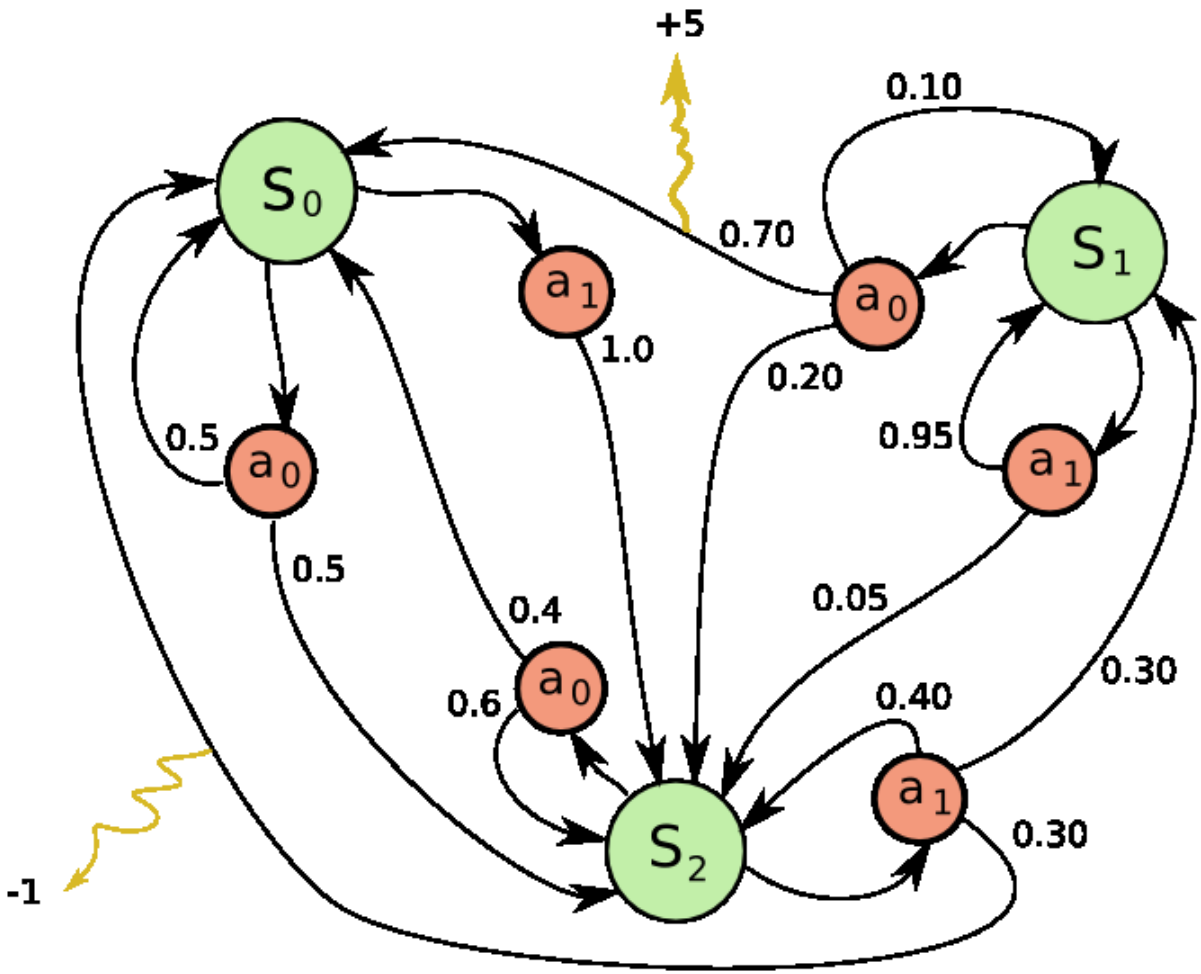
1. State space, $\mathcal{S}$
2. Action space, $\mathcal{A}$
3. Reward space, $\mathcal{R}$
4. Transition distribution, $\rho(s' \mid s, a)$. Given a previous state $s$ and action $a$, environment gives $s'$.
5. Reward function $r(s, a) \in \mathcal{R}$.

**Markov Property:** $\rho(s' \mid s, a)$ depends only on $s, a$ not previous states!

# Markov Decision Process (MDP)

ML@B

## Example MDP

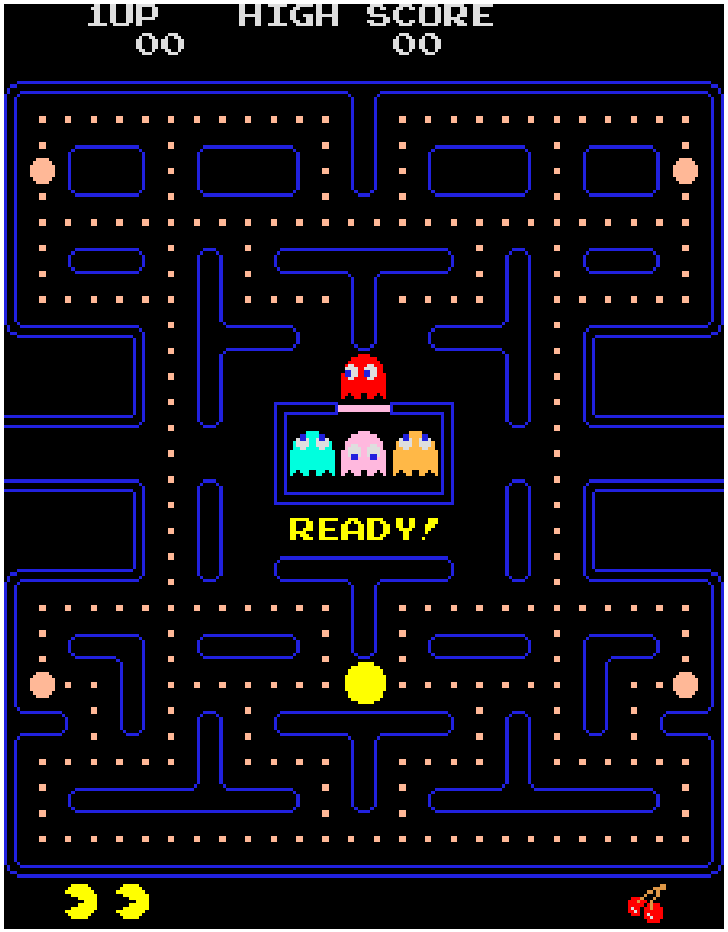


# Pacman as an MDP

ML@B



- $\mathcal{S} = \mathbb{R}^{256 \times 256}$, images as state space.
- $\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$, joystick as action space.
- $r(s_t, a_t) =$ change in score.
- $\rho(s_{t+1} \mid s_t, a_t) =$ next frame of game after joystick action $a_t$.



# Policies/Agents

ML@B

**Two different types of agents**

- Deterministic policy $a = \pi(s)$ acts in $E$.
- Stochastic policy $a \sim \pi(a|s)$ gives a probability distibution over actions.

**Policy Trajectories**

$$s_1 \xrightarrow{\pi} a_1 \xrightarrow{\rho,r} s_2, r_2 \xrightarrow{\pi} a_2 \xrightarrow{\rho,r} \cdots$$

# Value under a policy



The **state value** is a function of a given state for an agent $\pi$ defined as

$$V^{\pi}(s_t) = \mathbb{E}\left[\sum_{n=t+1}^{\infty} \gamma^n r(s_n, \pi(s_n))\right]$$

1. $\gamma$ is the discount factor
2. $\pi(s_n)$ is the action the agent $\pi$ makes after seeing state $s_n$.
3. $r(s_n, \pi(s_n))$ is the reward the agent gets from taking that action.

# Value under a policy



The **state-action value** for an agent $\pi$ is defined such that

$$Q^{\pi}(s_t, a_t) = \mathbb{E}\left[\underbrace{r(s_t, a_t)}_{\text{reward for } a_t} + V^{\pi}(s_t)\right]$$

- Given some state $s_t$, the *best* agent, $\pi^*$ is one that take action

$$a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a).$$

# Problems in Reinforcement Learning




- **Policy Optimization:** maximize the expected reward with respect to a policy $\pi$;

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$$

- **Policy Evaluation:** Given some fixed policy $\pi$ compute expected return.
  - Computing $Q^\pi$, $V^\pi$, and other expectations on policy rollout.
  - Lets us perform policy optimization!

# Assorted Algorithms

We'll go over:

- Behavioral Cloning
- Q-Learning
- Policy Iteration

Learn at home:

- Value iteration
- Temporal Difference Methods
- Inverse Reinforcement Learning.

# Behavioral Cloning



**Behavioral Cloning:** Supervised learning in MDPs using and expert agent expert $\pi^*$!

# Behavioral Cloning

ML@B

**Behavioral Cloning:** Supervised learning in MDPs using and expert agent expert $\pi^*$!

Given expert examples $\mathcal{D} = (s_t, a_t = \pi^*(s_t))$ and a model $\pi_\theta$ find $\theta^*$ st

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(a_t, \pi_\theta(s_t)).$$

where $\mathcal{L}$ is some loss function.

# Behavioral Cloning

ML@B

**Behavioral Cloning:** Supervised learning in MDPs using and expert agent expert $\pi^*$!

Given expert examples $\mathcal{D} = (s_t, a_t = \pi^*(s_t))$ and a model $\pi_\theta$ find $\theta^*$ st

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(a_t, \pi_\theta(s_t)).$$

where $\mathcal{L}$ is some loss function.

- Show, don't tell!

# Behavioral Cloning



**Behavioral Cloning:** Supervised learning in MDPs using and expert agent expert $\pi^*$!

Given expert examples $\mathcal{D} = (s_t, a_t = \pi^*(s_t))$ and a model $\pi_\theta$ find $\theta^*$ st

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(a_t, \pi_\theta(s_t)).$$

where $\mathcal{L}$ is some loss function.

- Show, don't tell!
- No complicated machinery, just standard ML.

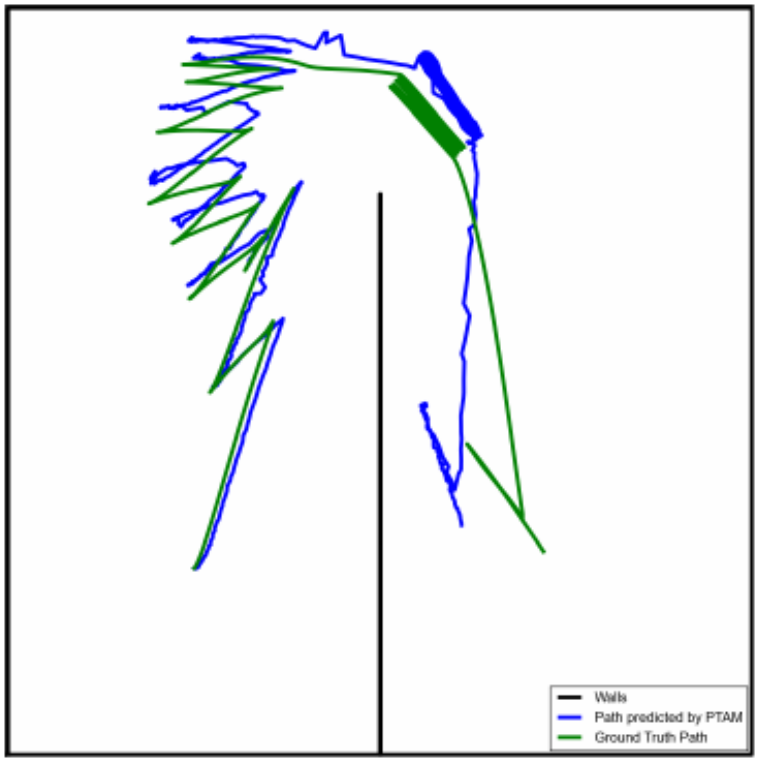
# Behavioral Cloning

ML@B

## Issue: Compounding Error

Given some irreducible error $\epsilon = 0.001$

- $\mathcal{L}(a_0, \pi_\theta(s_0)) = \epsilon$
- $\mathcal{L}(a_1, \pi_\theta(s_1)) = 2\epsilon$
- $\mathcal{L}(a_2, \pi_\theta(s_2)) = 3\epsilon$
- $\mathcal{L}(a_3, \pi_\theta(s_3)) = 4\epsilon$
- $\mathcal{L}(a_4, \pi_\theta(s_4)) = 5\epsilon$



# Behavioral Cloning

ML@B

## Issue: Distribution Mismatch

- States expert dataset $\mathcal{D}$ generated by $\pi^*$ have different distribution than those generated by $\pi_\theta$.
  $\implies$ No self correction.

# Behavioral Cloning

ML@B

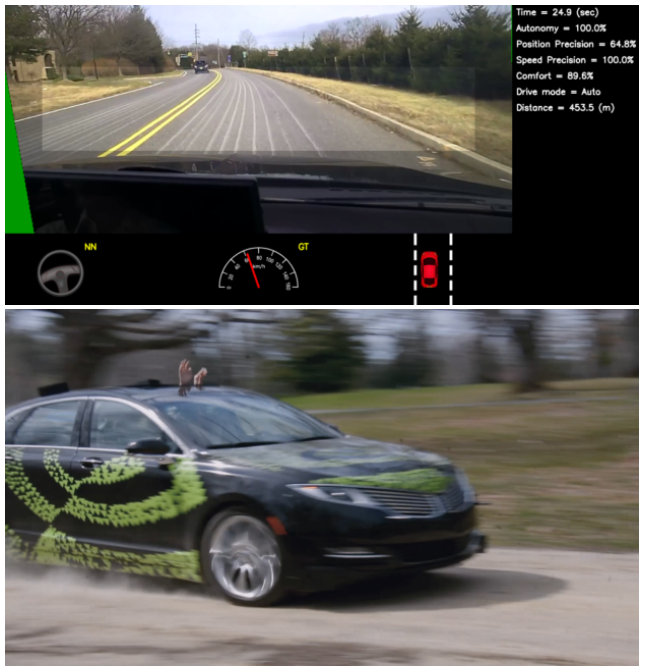
**Issue: Distribution Mismatch**

- States expert dataset $\mathcal{D}$ generated by $\pi^*$ have different distribution than those generated by $\pi_\theta$.

  $\implies$ No self correction.

**Solution: DAgger**.

- Do BC on $\mathcal{D}$ and generate $E_0$ states generated by $\pi_\theta$.
- Label $E_0$ with expert level actions and add to $\mathcal{D}$.



## Goals of Q-learning

1. Approximate $Q^{\pi^*}$, the $Q$ function of the optimal agent, as $Q(s_t, a_t)$.

# Q-Learning (State-action Value Iteration)




## Goals of Q-learning

1. Approximate $Q^{\pi^*}$, the $Q$ function of the optimal agent, as $Q(s_t, a_t)$.
2. Using $Q$, find the agent, $\pi$, that best approximates the optimal agent, $\pi^*$.

# Q-Learning (State-action Value Iteration)




**How do we define best?**

# Q-Learning (State-action Value Iteration)




**How do we define best?**

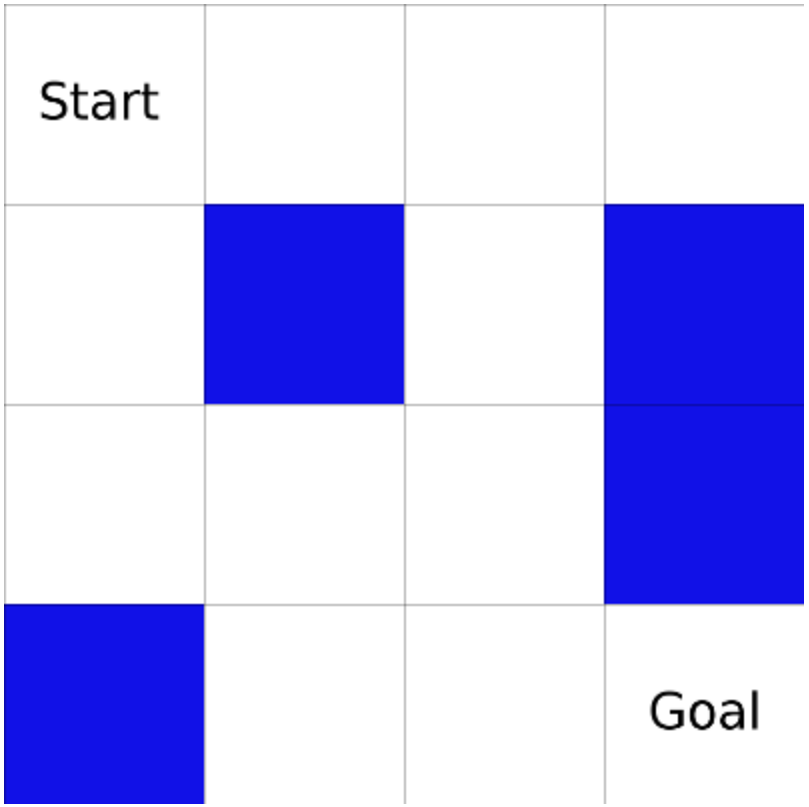
Given some state $s_t$, the **best** agent, $\pi^*$ is one that takes action

$$a_t = \arg\max_a Q(s_t, a).$$

# Q-Learning (State-action Value Iteration)



**An example: Frozen Lake Problem**



# Q-Learning (State-action Value Iteration)




- $100$ reward for reaching the goal
- $0$ otherwise

**How do we keep track of this long term reward?**

# Q-Learning (State-action Value Iteration)




- $100$ reward for reaching the goal
- $0$ otherwise

**How do we keep track of this long term reward?**

**$Q$ function**

# Q-Learning (State-action Value Iteration)




**How do we actually calculate the $Q$ function?**

# Q-Learning (State-action Value Iteration)




**How do we actually calculate the $Q$ function?**

**The Bellman Equation.**

# Q-Learning (State-action Value Iteration)




**How do we actually calculate the $Q$ function?**

**The Bellman Equation.**

$$Q^{\pi}(s_t, a_t) = r_t + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))$$

# Q-Learning (State-action Value Iteration)




**One $Q$-Learning Algorithm: Tabular Q-Learning**

- Explore the environment
- On the way, use the Bellman equation to store a table of expected future reward ($Q$) for each state-action pair.
- Use this table to pick the best possible action for any given state.

# Q-Learning (State-action Value Iteration)




**An example update for Frozen Lake.**

Suppose our stored $Q$ table looks like so:

| Up | Down | Left | Right |
|---|---|---|---|
| 0 | 65 | 0 | 40 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 50 | 75 | 30 | 20 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

# Q-Learning (State-action Value Iteration)




**An example update for Frozen Lake.**

Then suppose our agent moves **down** from the starting square

# Q-Learning (State-action Value Iteration)

ML@B

**An example update for Frozen Lake.**

Then we update using the Bellman equation.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha(r_t + \gamma(\max_a Q(s_t, a) - Q(s_t, a_t))$$

| Up | Down | Left | Right |
|---|---|---|---|
| 0 | 65 | 0 | 40 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 50 | 75 | 30 | 20 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

# Q-Learning (State-action Value Iteration)




**An example update for Frozen Lake.**

The table now looks like so:

| Up | Down | Left | Right |
|---|---|---|---|
| 0 | 70 | 0 | 40 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 50 | 75 | 30 | 20 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

# Policy Iteration



**Policy Iteration:** Given access to the MDP, use policy evaluation to iteratively serach for better policies!

- Choose a policy at random, $\pi$.

# Policy Iteration

ML@B

**Policy Iteration:** Given access to the MDP, use policy evaluation to iteratively serach for better policies!

- Choose a policy at random, $\pi$.
- Alternate between
  - Evaluate policy $\pi \rightarrow V^{\pi}$.

# Policy Iteration



**Policy Iteration:** Given access to the MDP, use policy evaluation to iteratively serach for better policies!

- Choose a policy at random, $\pi$.
- Alternate between
  - Evaluate policy $\pi \to V^\pi$.
  - Set new policy to be greedy policy for $V^\pi$

$$\begin{aligned}\pi(s) &:= \operatorname*{argmax}_a \mathbb{E}\left[R(s,a) + \gamma V^\pi(s')\right] \\ &:= \operatorname*{argmax}_a Q^\pi(s,a)\end{aligned}$$

# Policy Iteration

ML@B

**Policy Iteration:** Given access to the MDP, use policy evaluation to iteratively serach for better policies!

- Choose a policy at random, $\pi$.
- Alternate between
  - Evaluate policy $\pi \to V^\pi$.
  - Set new policy to be greedy policy for $V^\pi$

$$\begin{aligned}\pi(s) &:= \operatorname*{argmax}_a \mathbb{E}\left[R(s,a) + \gamma V^\pi(s')\right] \\ &:= \operatorname*{argmax}_a Q^\pi(s,a)\end{aligned}$$

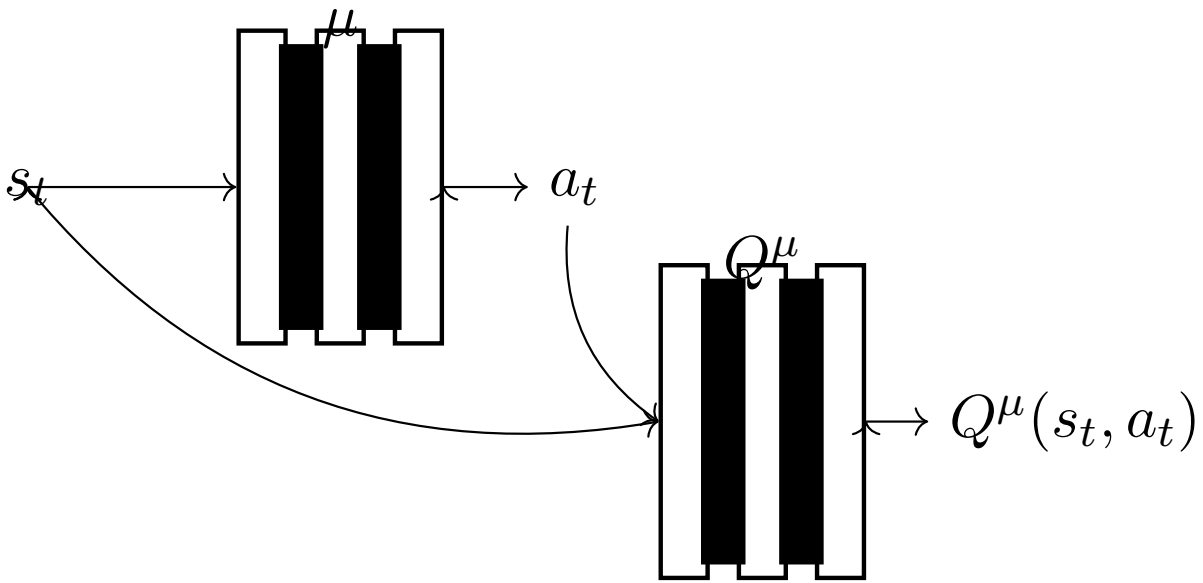
- Learn $Q^\pi$ using Q-learning without $\operatorname{argmax}$.

# Policy Iteration

ML@B

## Deep Determisitic Policy Gradient

1. Actor neural network $\mu : \mathcal{S} \rightarrow \mathcal{A}$
2. Critic network $Q^{\mu} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
3. Performance of $\mu$ is $Q^{\mu}(s_t, \mu(s_t))$. **Maximize performance!** $\nabla_W Q^{\mu}(s_t, a_t) = \nabla_a Q^{\mu}(s_t, a) \cdot \nabla_W \mu(s_t)$



# Questions?