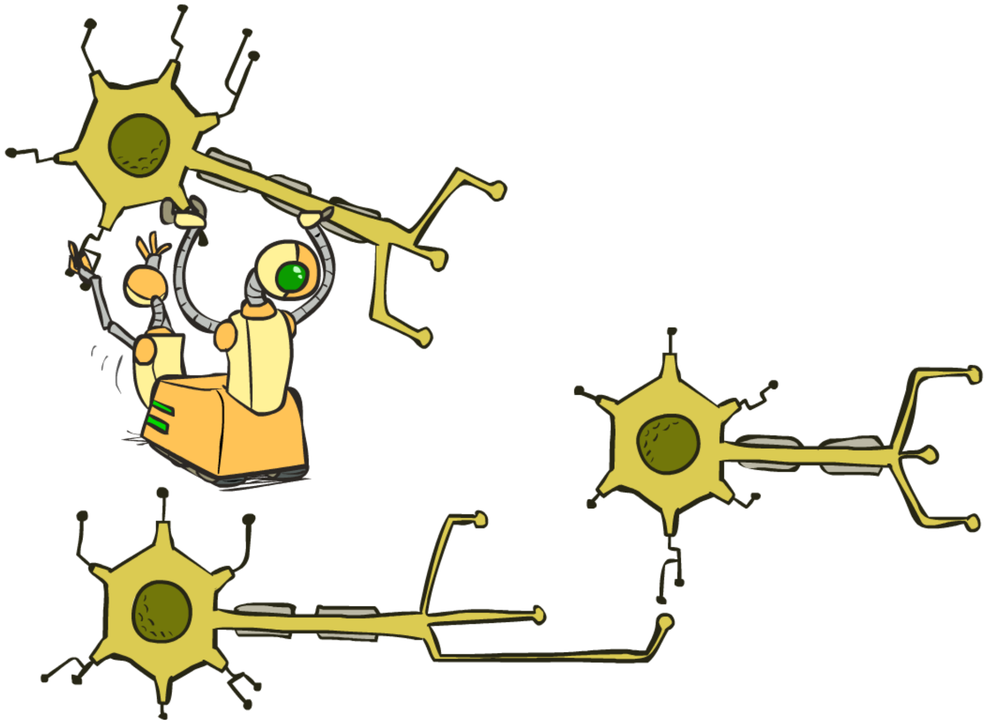
CS 188: Artificial Intelligence

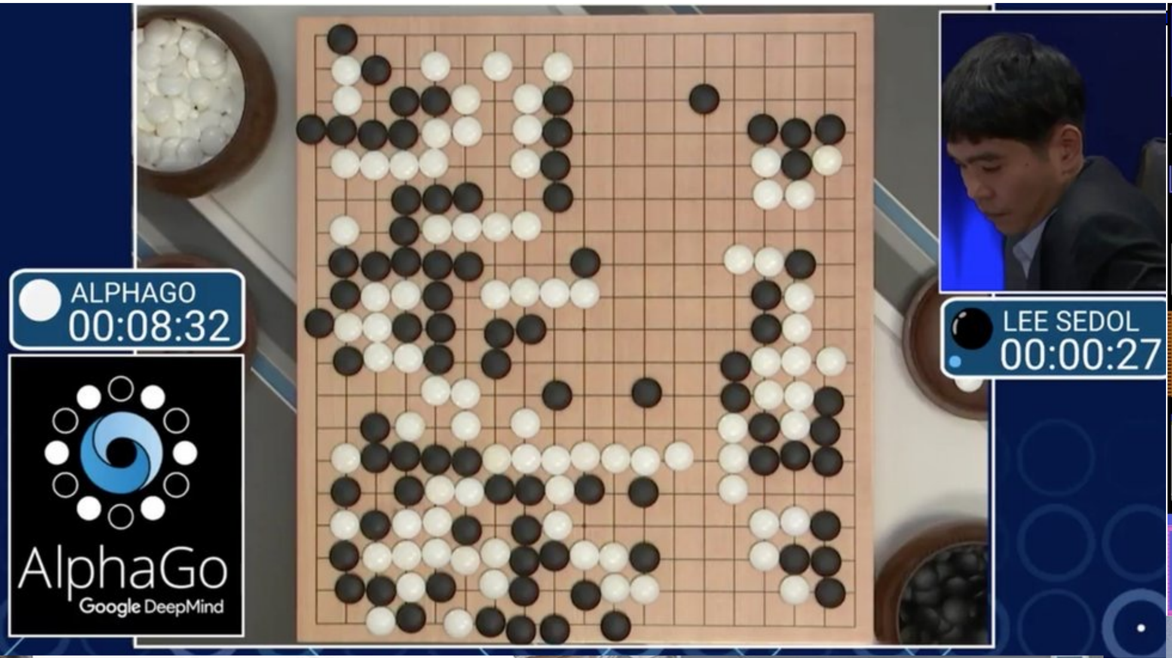
Special Topics: NLP/CV/RL

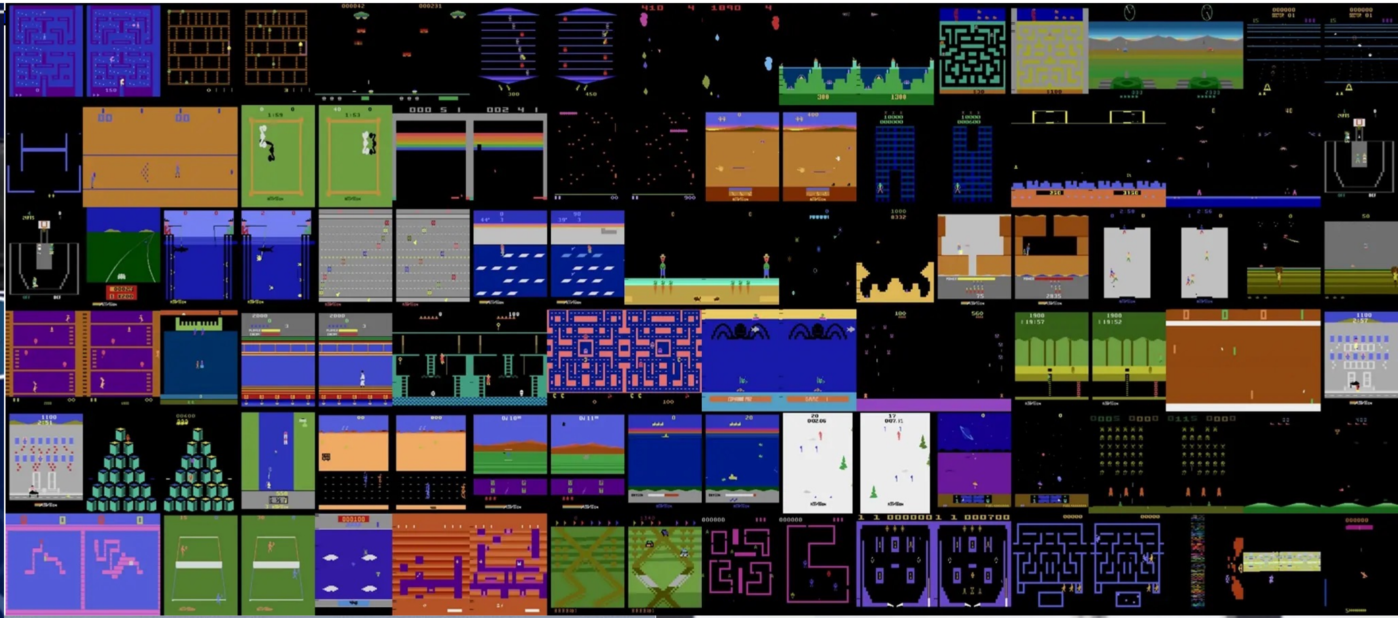


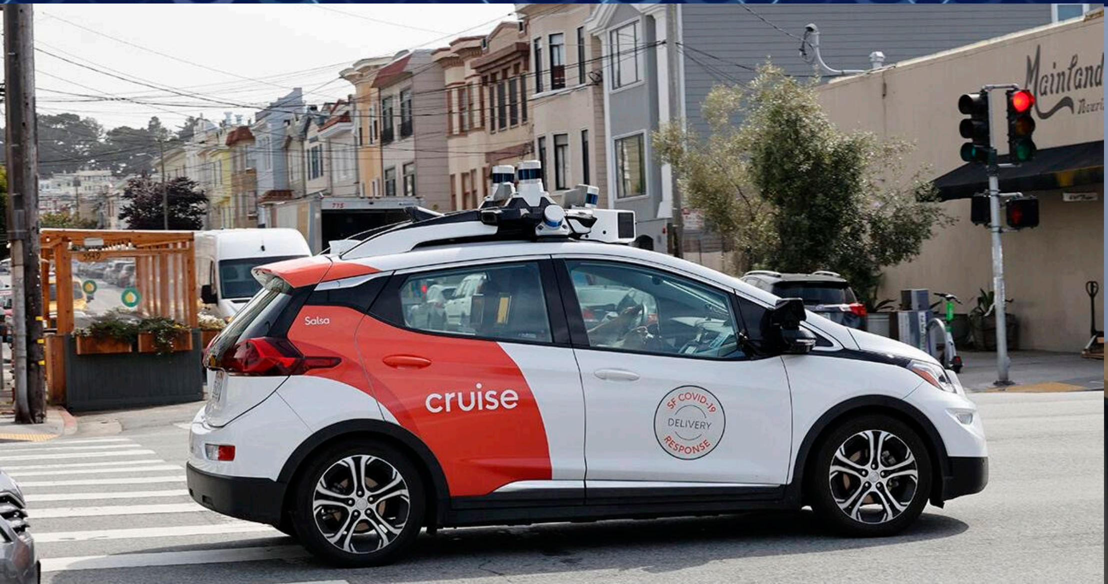
Instructor: Nicholas Tomlin

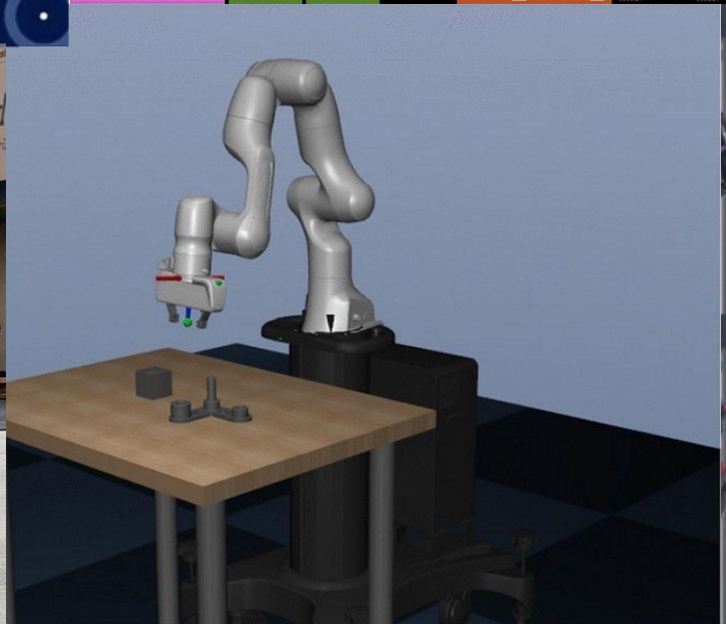
[Slides courtesy of Dan Klein, Abigail See, Greg Durrett, Yejin Choi, John DeNero, Eric Wallace, Kevin Lin, Fei-Fei Li, Sergey Levine, Pieter Abbeel, and many others]

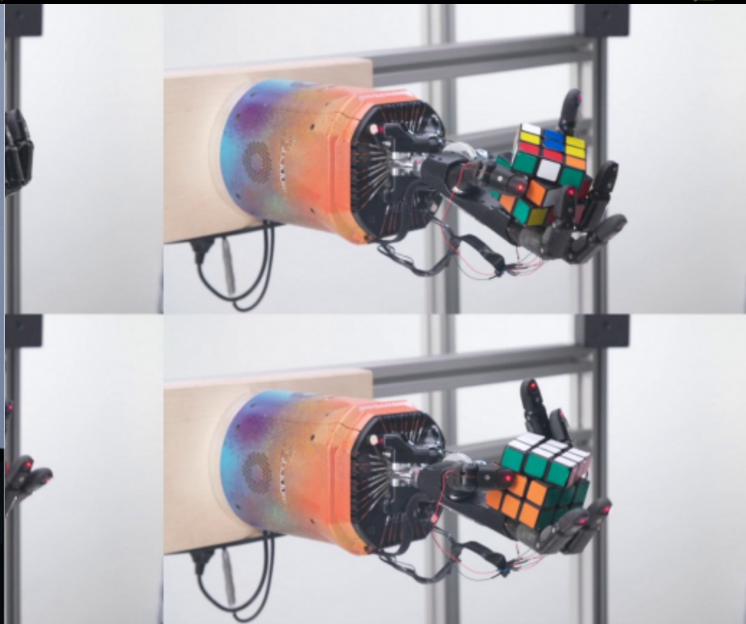
What tasks do we care about?



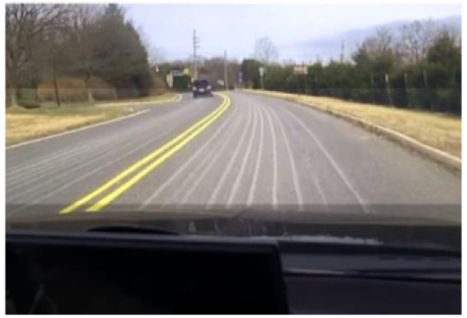




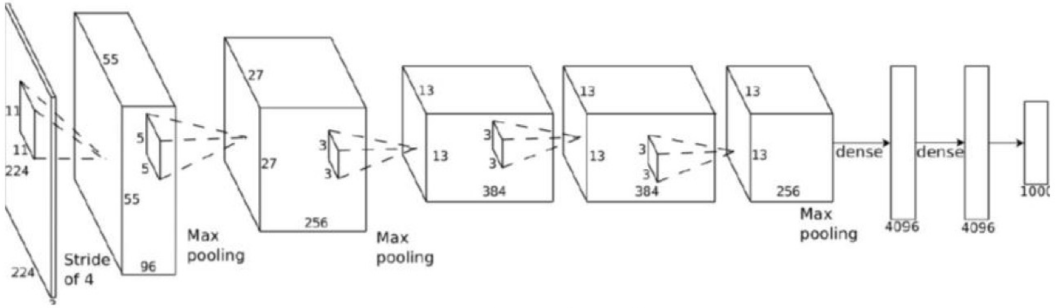




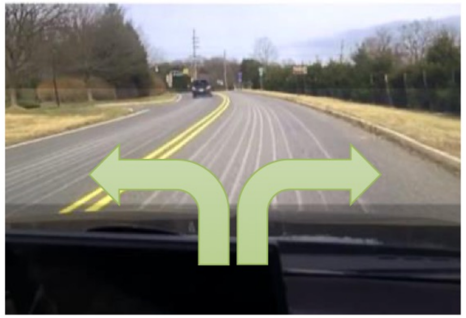
Imitation Learning



$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$



$\mathbf{a}_t$

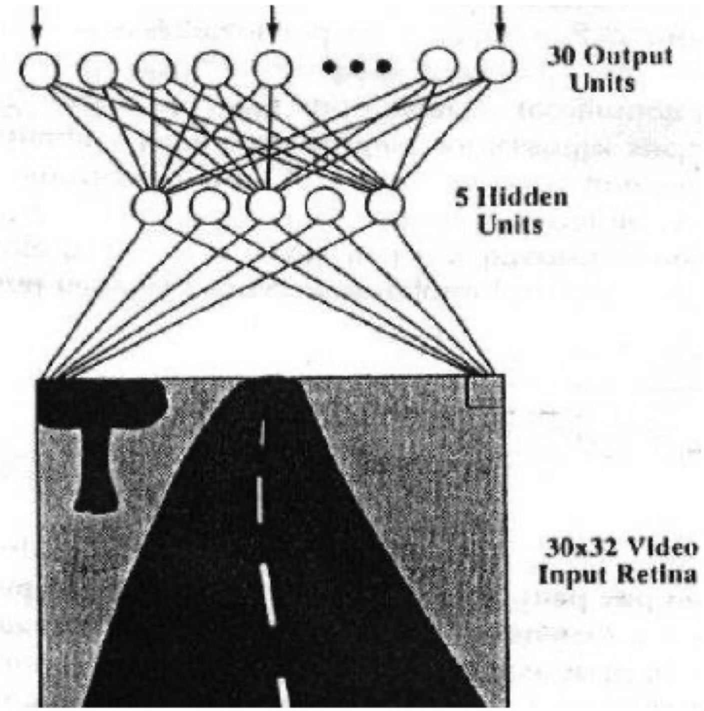
$\mathbf{o}_t$

$\mathbf{a}_t$

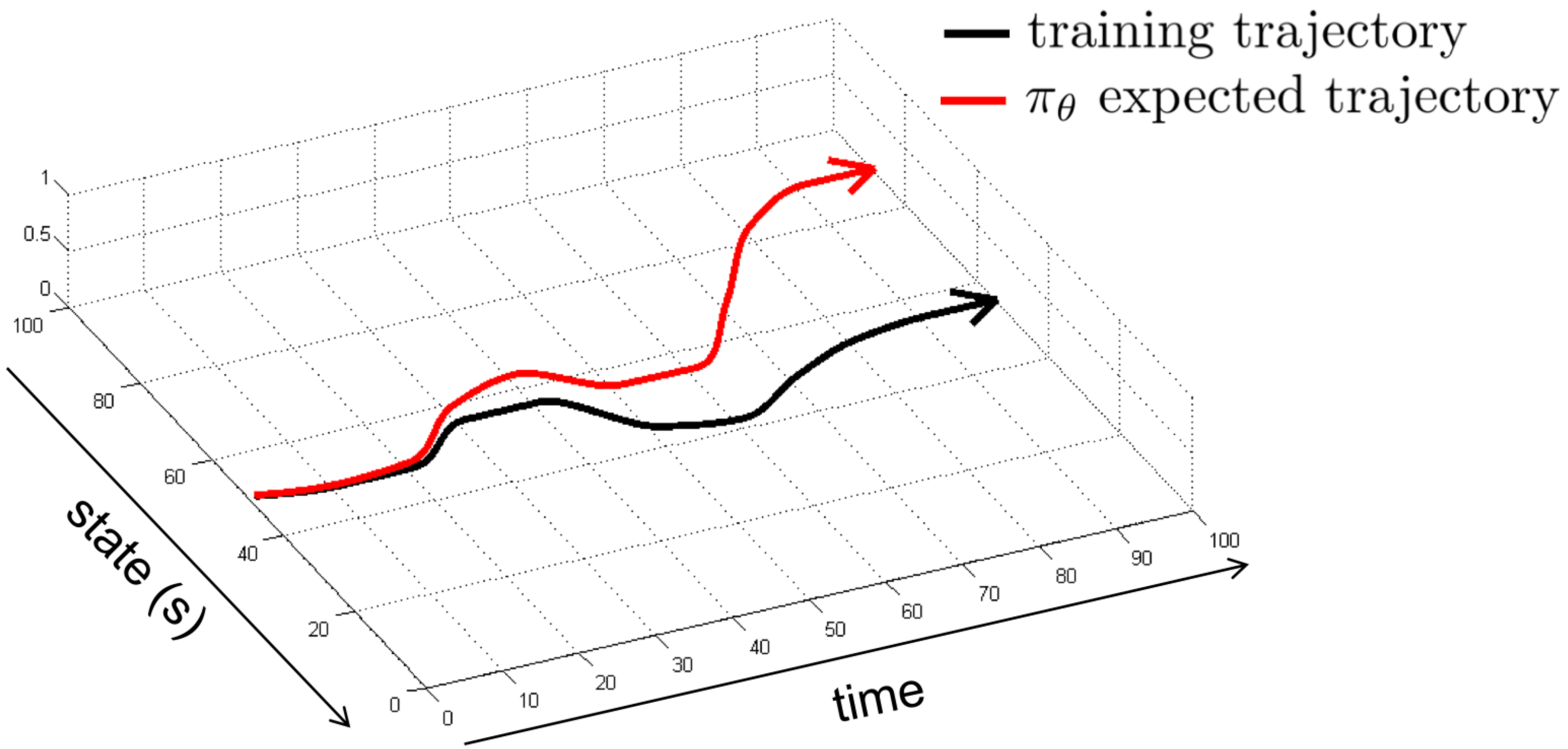
→ training data → supervised learning

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

ALVINN: **A**utonomous **L**and **V**ehicle **I**n a **N**eural **N**etwork

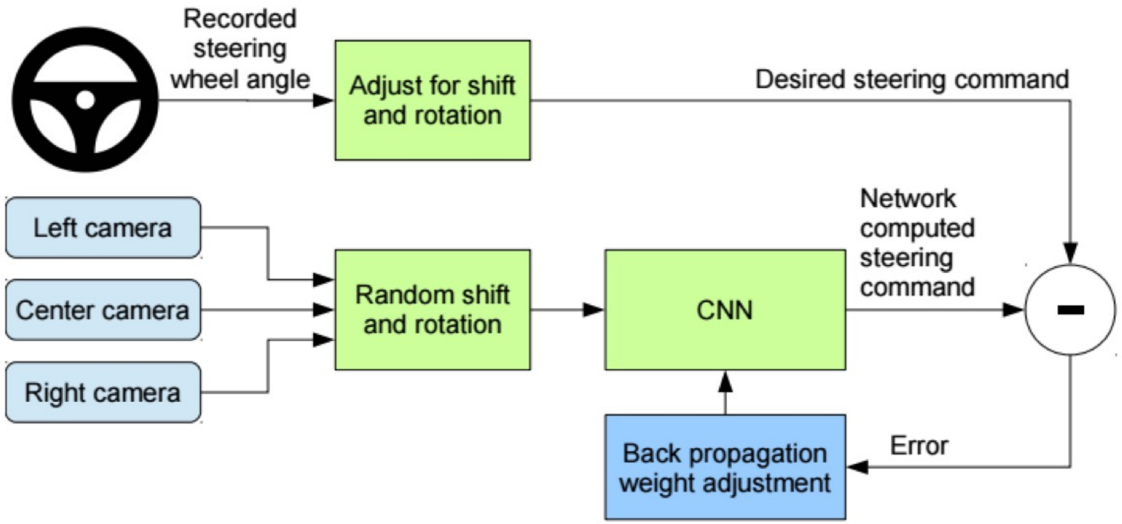


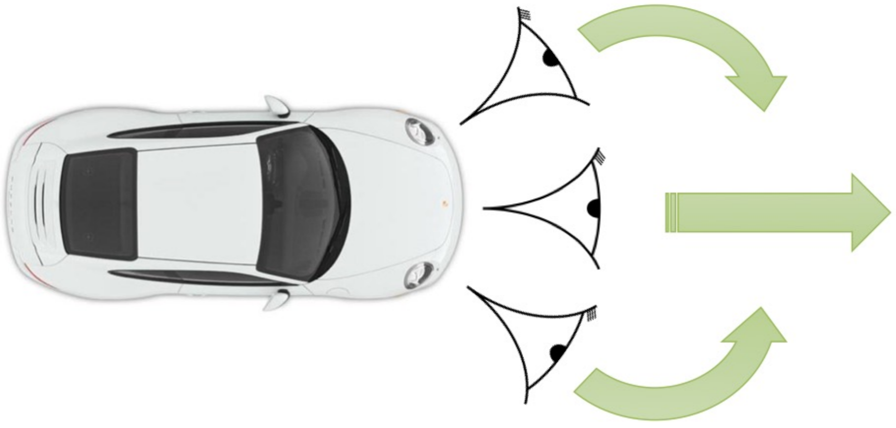
Distributional Drift



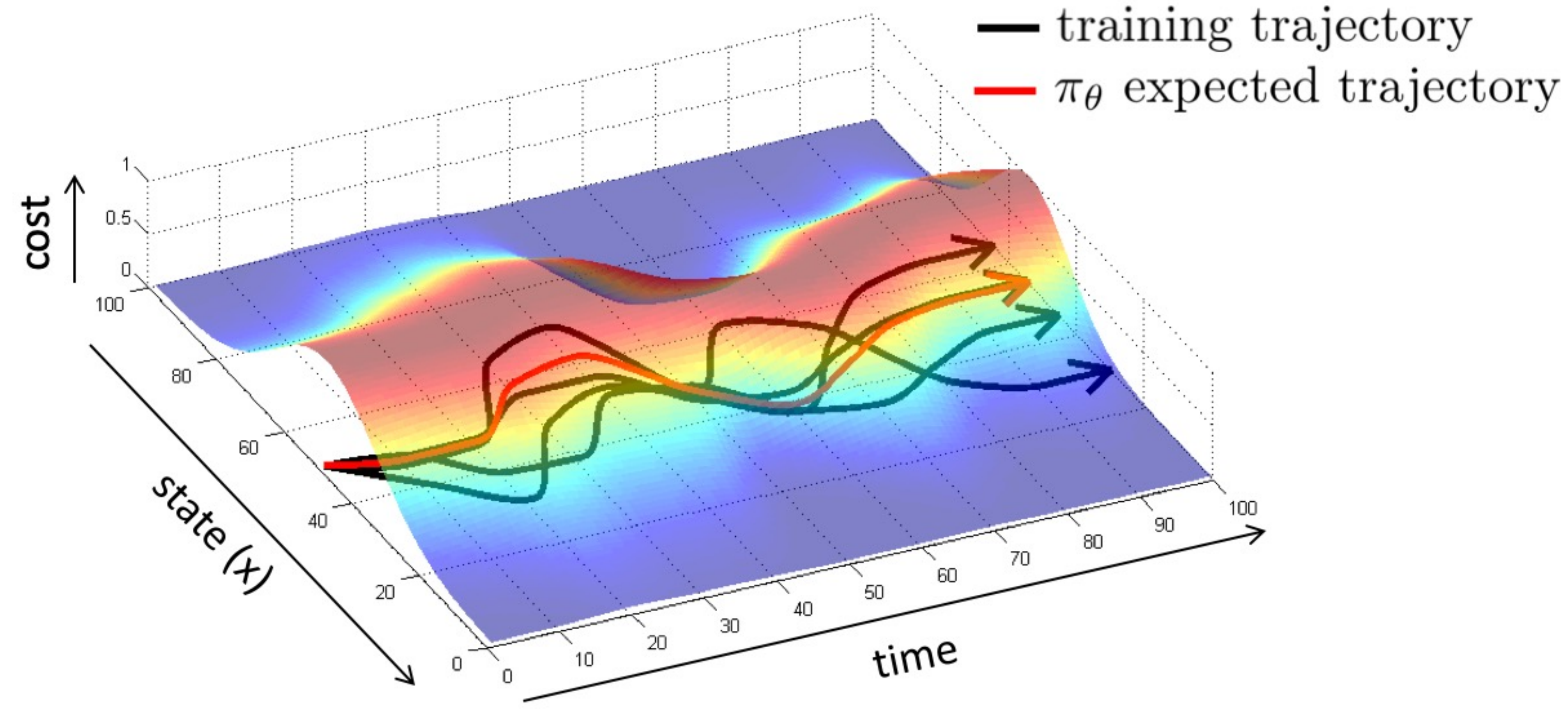
Modern Approach to Autonomous Driving

Modern Approach to Autonomous Driving

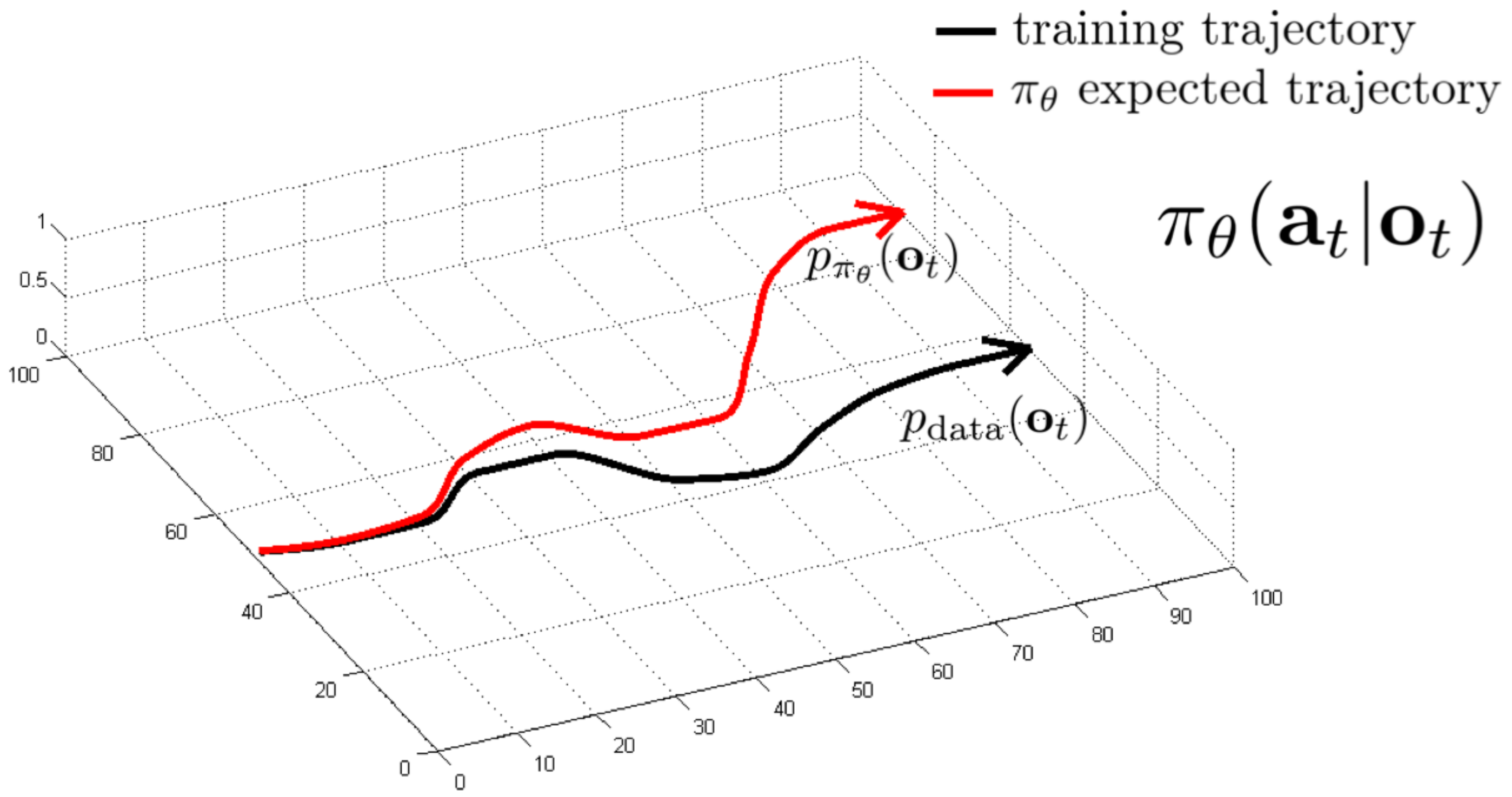




Avoiding Compounding Errors (Stability)



Avoiding Distributional Drift



$$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$$

can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?

DAgger: **D**ataset **A**ggregation (Ross, et al. 2011)

can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?

idea: instead of being clever about $p_{\pi_\theta}(\mathbf{o}_t)$, be clever about $p_{\text{data}}(\mathbf{o}_t)$!

DAgger: **D**ataset **A**ggregation

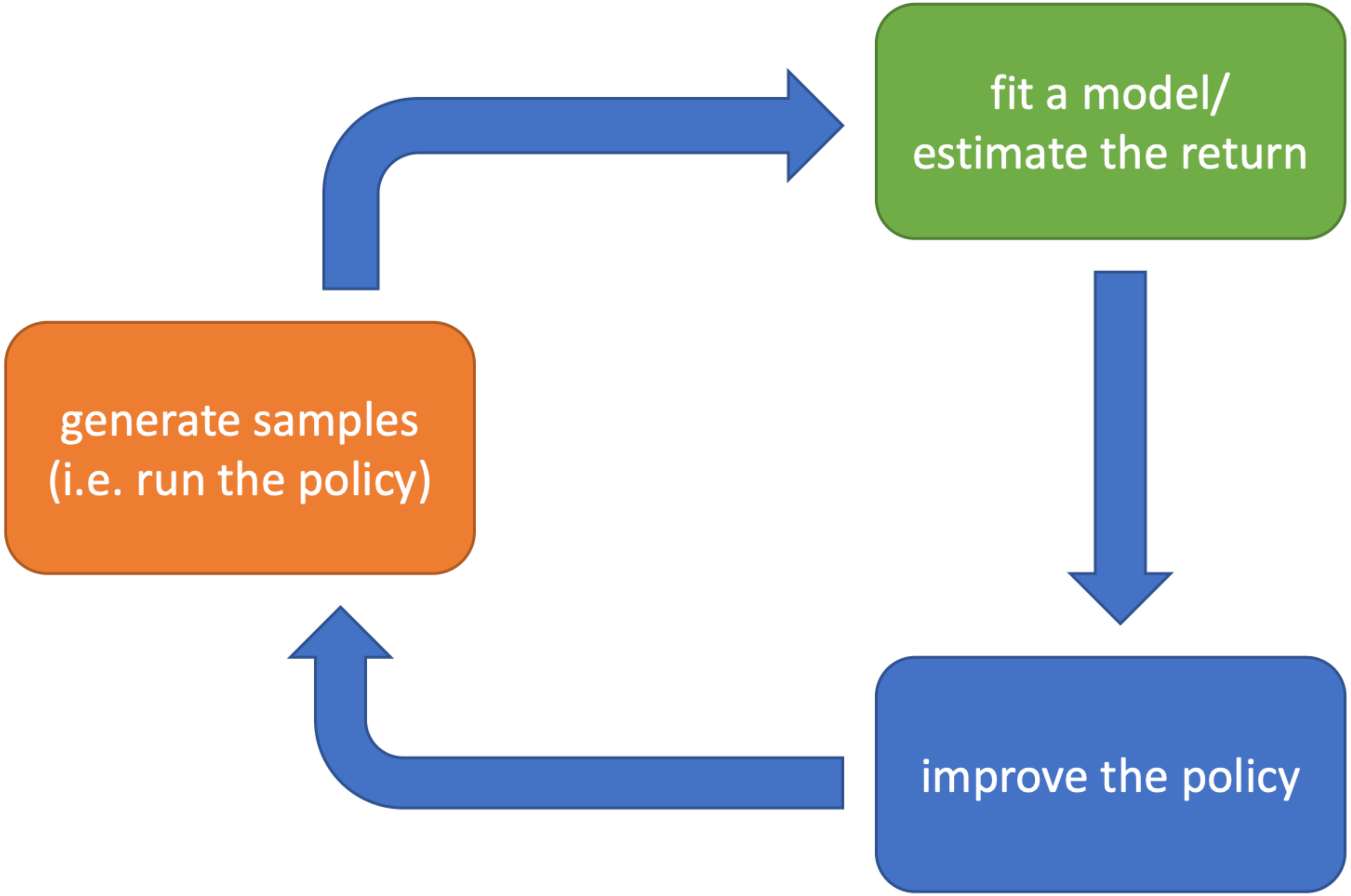
goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\text{data}}(\mathbf{o}_t)$

how? just run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

but need labels $\mathbf{a}_t$!

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \dots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \dots, \mathbf{o}_M\}$
3. $\boxed{\text{Ask human to label } \mathcal{D}_\pi \text{ with actions } \mathbf{a}_t}$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Reinforcement Learning



Recall: Map of Reinforcement Learning

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

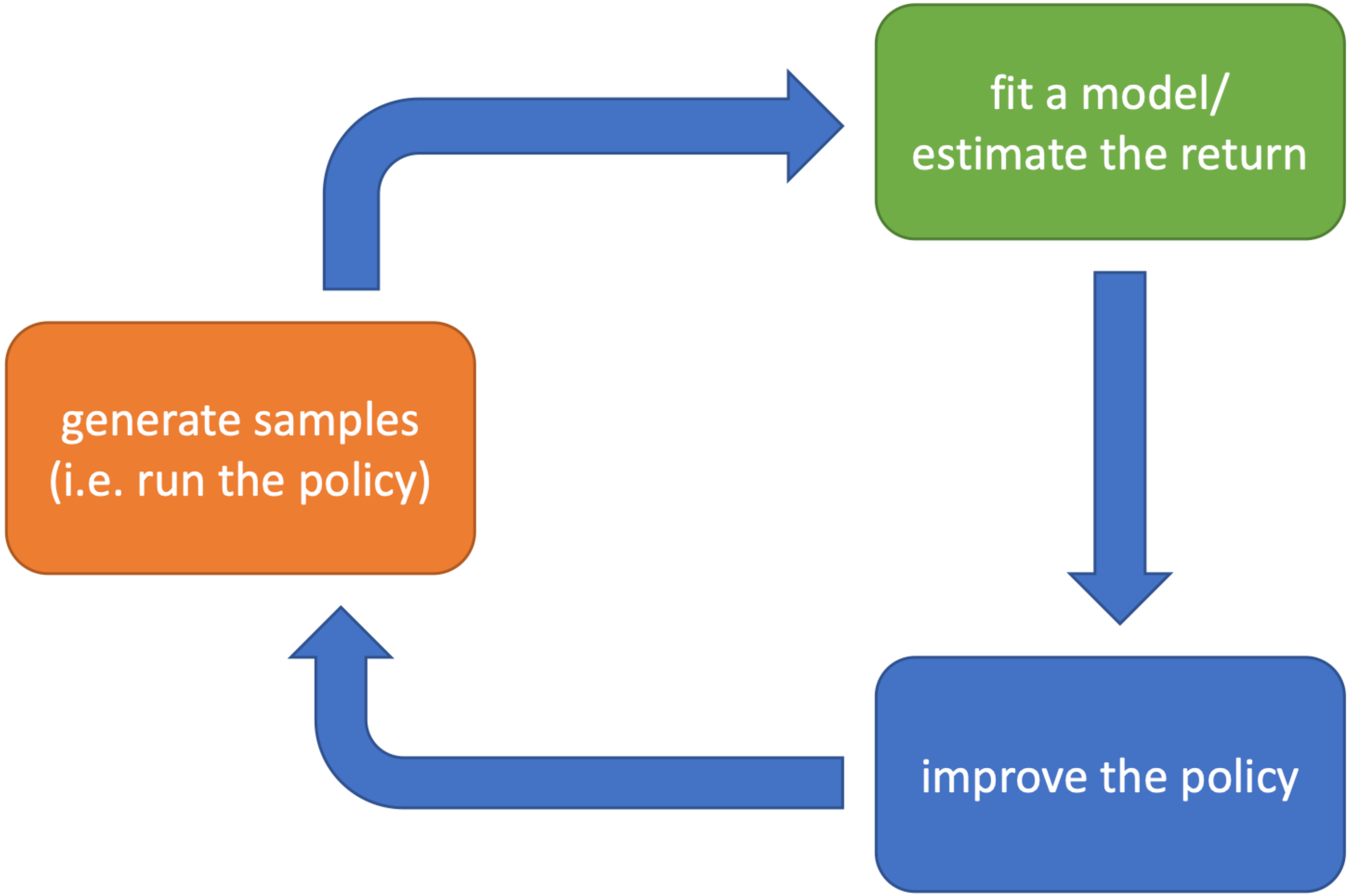
Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	Value Learning

Map of Reinforcement Learning

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

- Policy gradient: directly differentiate the above equation
- Value-based: estimate value function or Q-function of the optimal policy directly (but no explicit policy)
- Actor-critic: estimate value function or Q-function of the *current* policy, and use it to improve the policy
- Model-based RL: estimate the transition model, and then:
  - Use it for planning
  - Use it to improve a policy

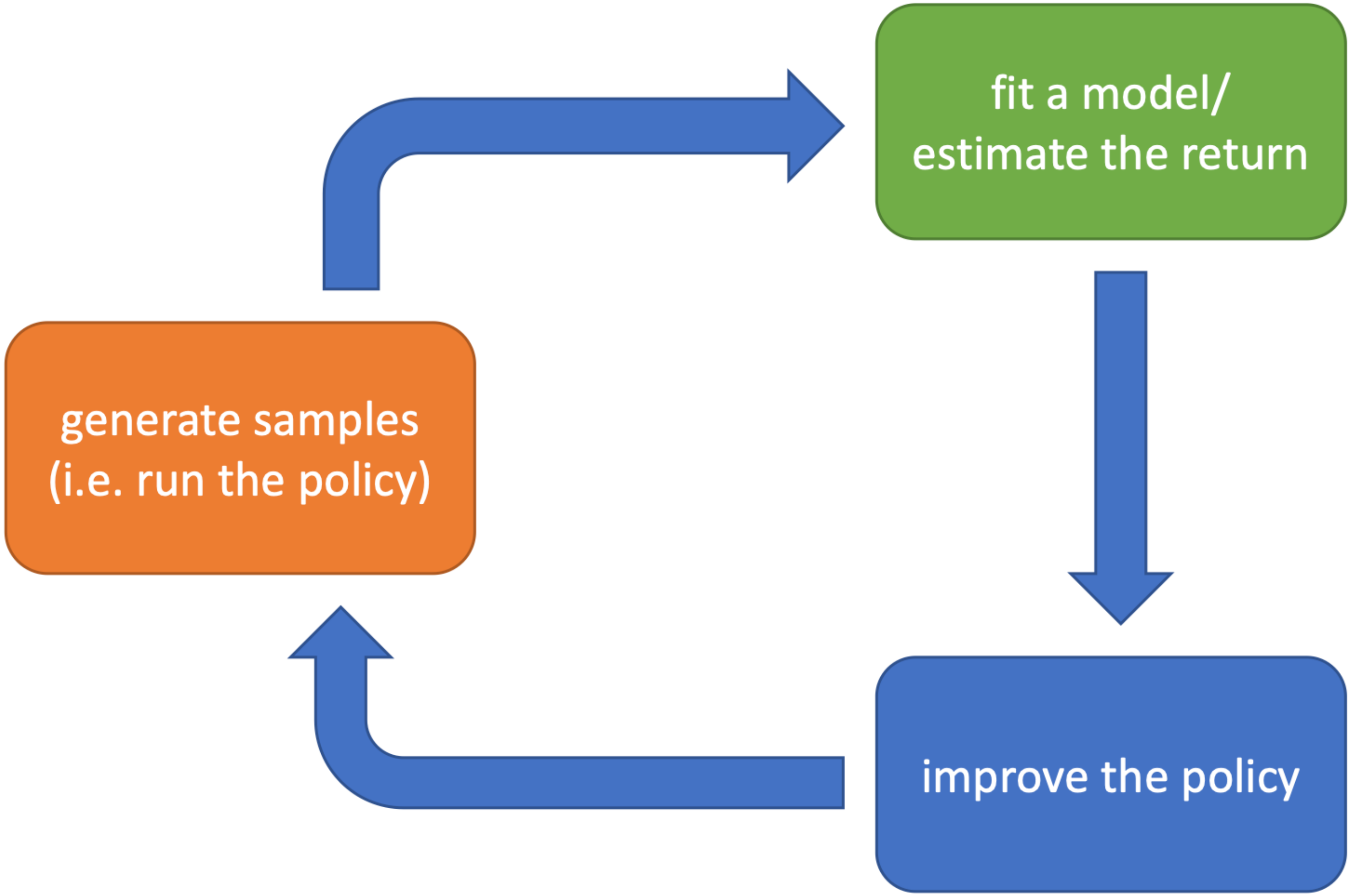
Policy Gradient



evaluate returns $R_\tau = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

$\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$

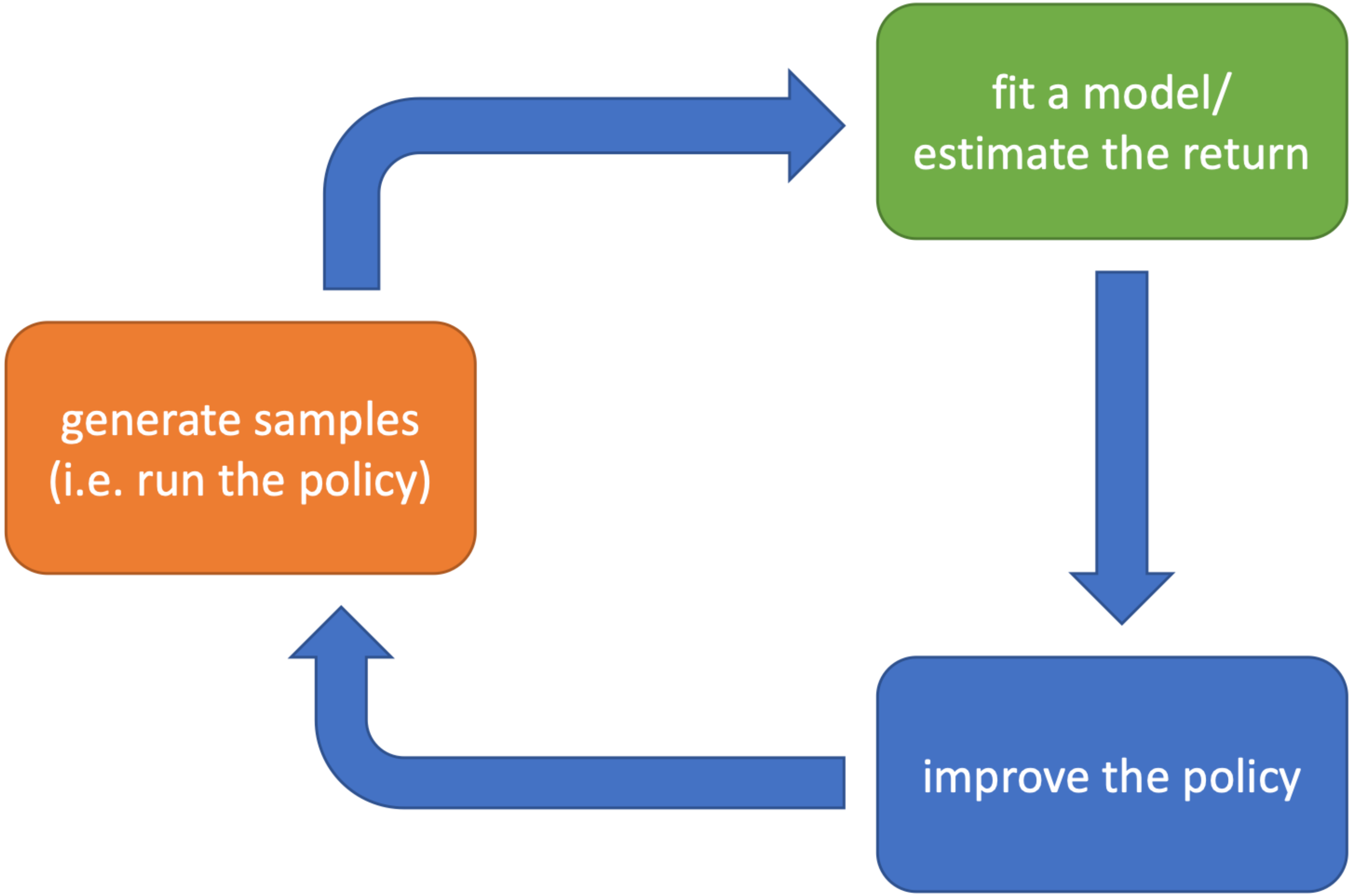
Value Function-Based Approaches



fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

set $\pi(\mathbf{s}) = \arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

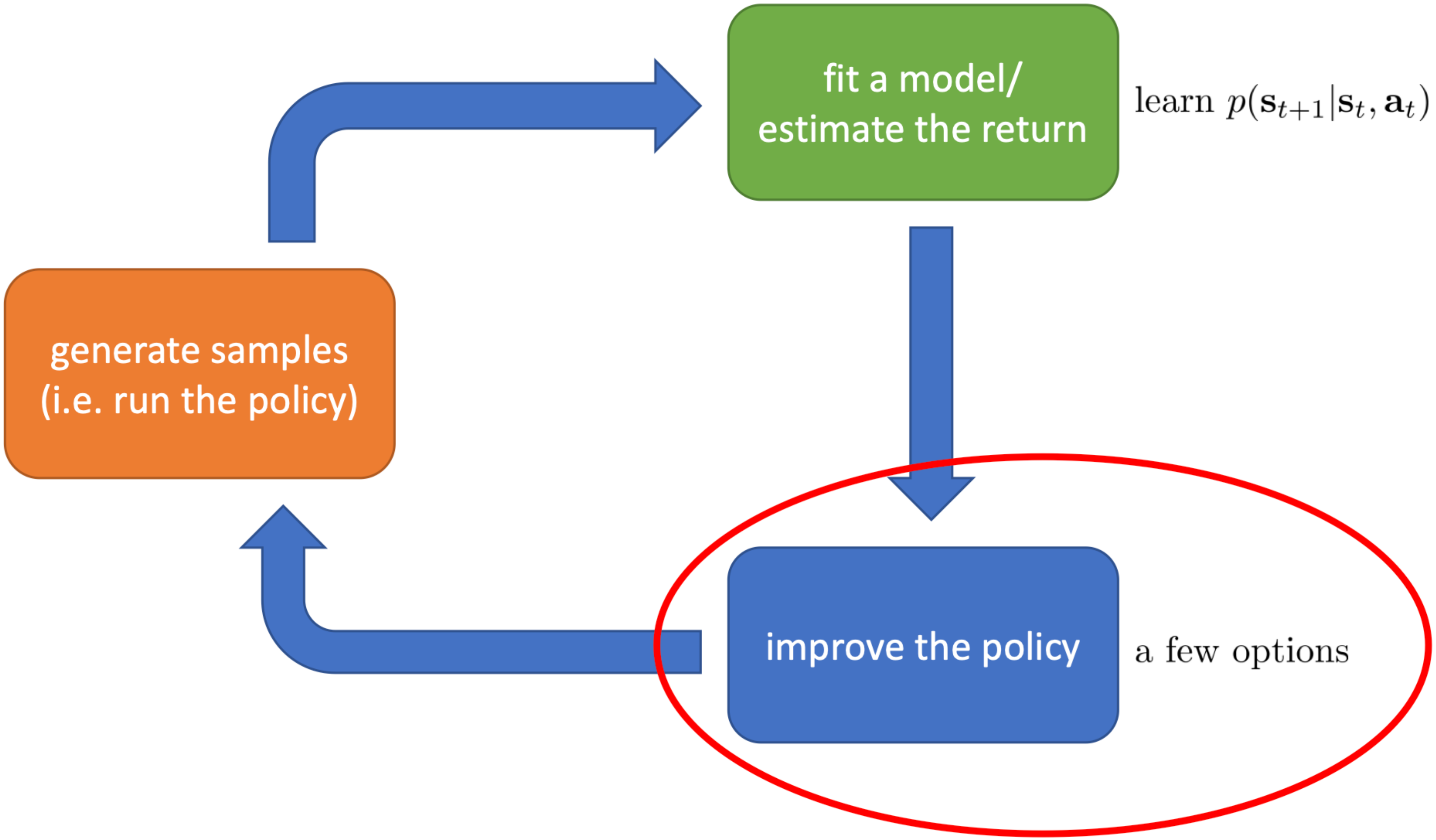
Actor-Critic: Value Functions + Policy Gradients



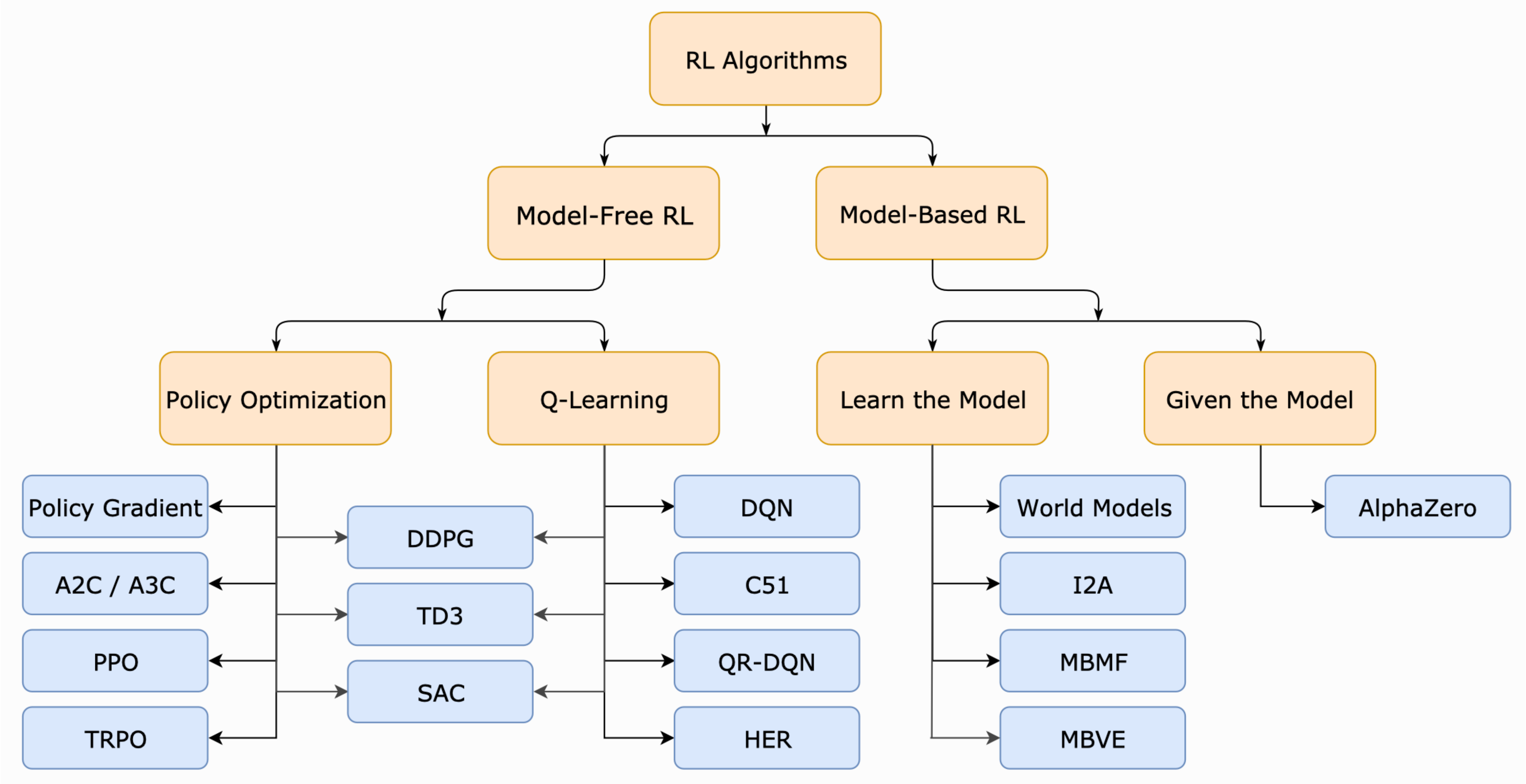
fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

$$\theta \leftarrow \theta + \alpha \nabla_\theta E[Q(\mathbf{s}_t, \mathbf{a}_t)]$$

Model-Based Reinforcement Learning



Map of Reinforcement Learning

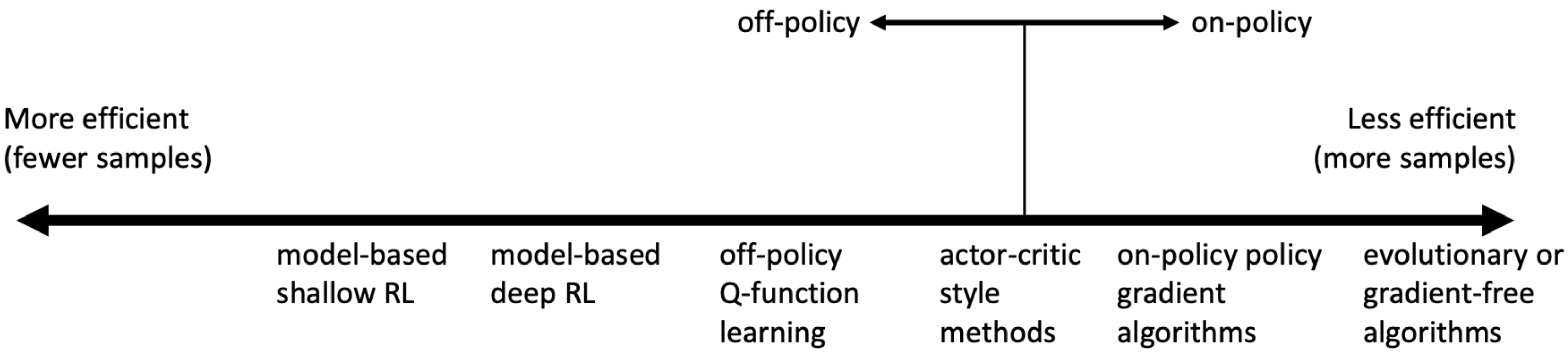


Why so many options?

- Different tradeoffs:
  - Sample efficiency
  - Stability and ease of use
- Different assumptions:
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?
- Different things are easy or hard in different settings:
  - Easy to represent the policy?
  - Easy to represent the model?

Comparison: Efficiency

- Sample efficiency = how many samples we need to get a good policy
- Most important question: is the algorithm on-policy or off-policy?
  - On-policy: each time the policy is changed, need to generate new samples
  - Off-policy: able to improve the policy without generating new samples



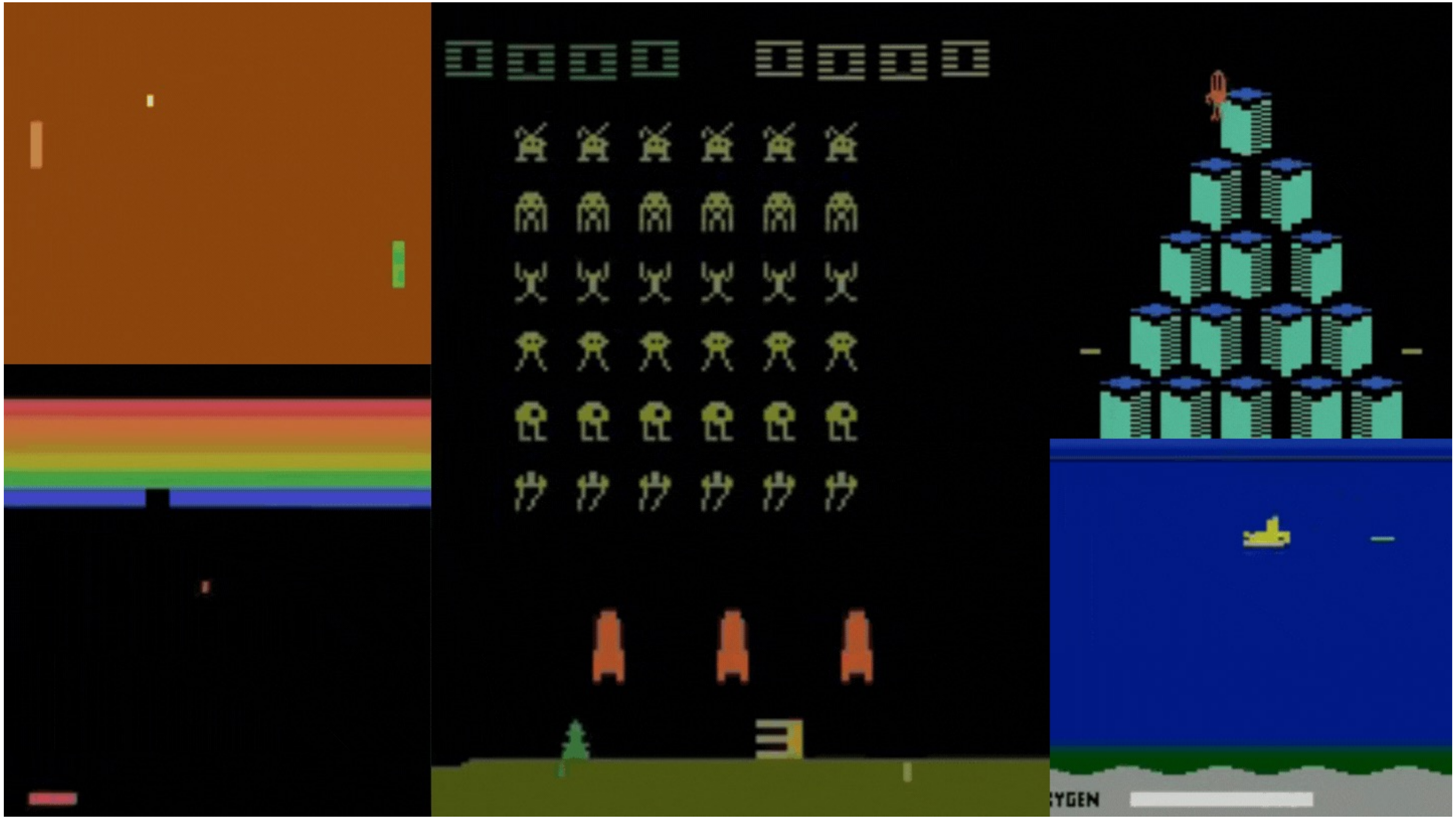
Comparison: Stability and Ease of Use

- Value function fitting:
  - At best, minimizes error of fit (“Bellman error”)
  - At worst, doesn’t optimize anything (often no guarantees with deep RL)
- Model-based RL:
  - Model minimizes error of fit (will converge)
  - No guarantee that better model = better policy
- Policy gradient:
  - The only approach that actually performs gradient descent on the true objective
  - In practice, often the least efficient!

Comparison: Assumptions

- Common assumption #1: full observability
  - Generally assumed by value function fitting methods
  - Can be mitigated by adding recurrence
- Common assumption #2: episodic learning
  - Often assumed by pure policy gradient methods
  - Assumed by some model-based RL methods
- Common assumption #3: continuity or smoothness
  - Assumed by some continuous value function learning methods
  - Often assumed by some model-based RL methods

Model-Free RL: Q-Learning



Recall: Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

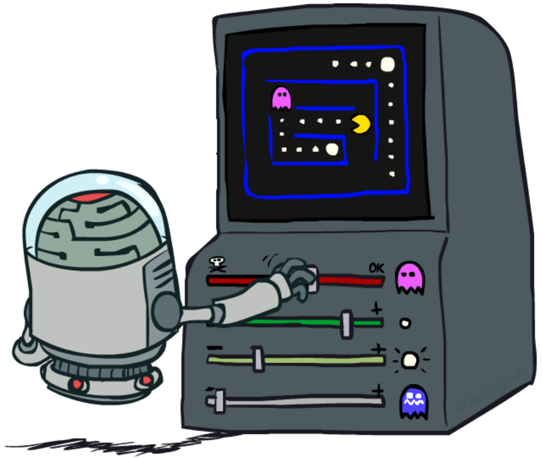
- Q-learning with linear Q-functions:

  transition $= (s, a, r, s')$

  difference $= \left[r + \gamma \max_{a'} Q(s', a') \right] - Q(s,a)$

  $Q(s,a) \leftarrow Q(s,a) + \alpha$ [difference] Exact Q's

  $w_i \leftarrow w_i + \alpha$ [difference] $f_i(s,a)$ Approximate Q's



- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Can perform update based on a single sample, or with multiple

Model-Free RL: DQN

Algorithm 1 Deep Q-learning with Experience Replay

Initialize $\boxed{\text{replay memory } \mathcal{D}}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    $\boxed{\text{With probability } \epsilon \text{ select a random action } a_t}$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    $\boxed{\text{Store transition } (\phi_t, a_t, r_t, \phi_{t+1}) \text{ in } \mathcal{D}}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    $\boxed{\text{Perform a gradient descent step on } (y_j - Q(\phi_j, a_j; \theta))^2}$ according to equation $\boxed{3}$
  end for
end for

Q-function is represented as a CNN

Model-Free RL: REINFORCE

REINFORCE algorithm:

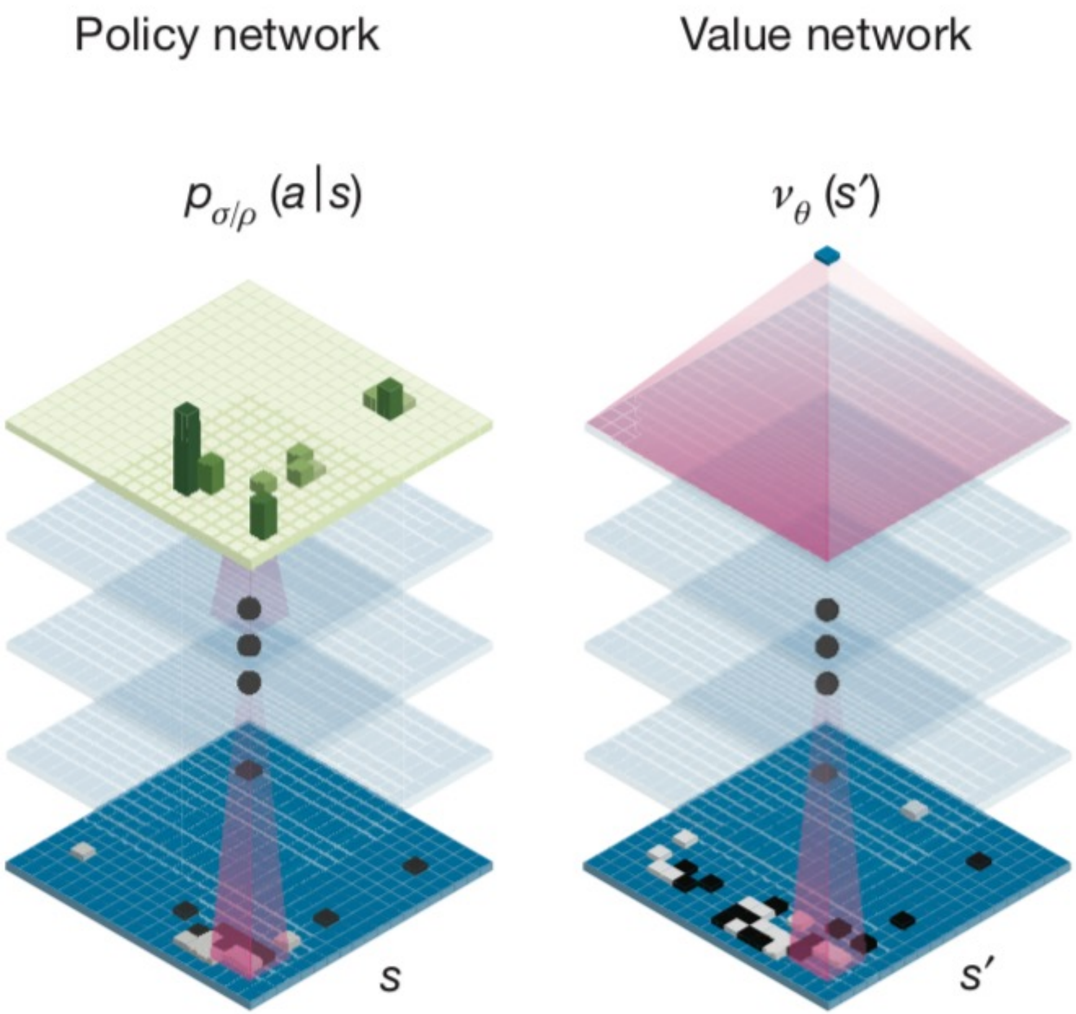
1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

- Inefficient: run the policy to get trajectories and then throw them away
- Gradient computations may be noisy (high variance)
- Practical considerations with batch sizes, learning rates, and optimizers

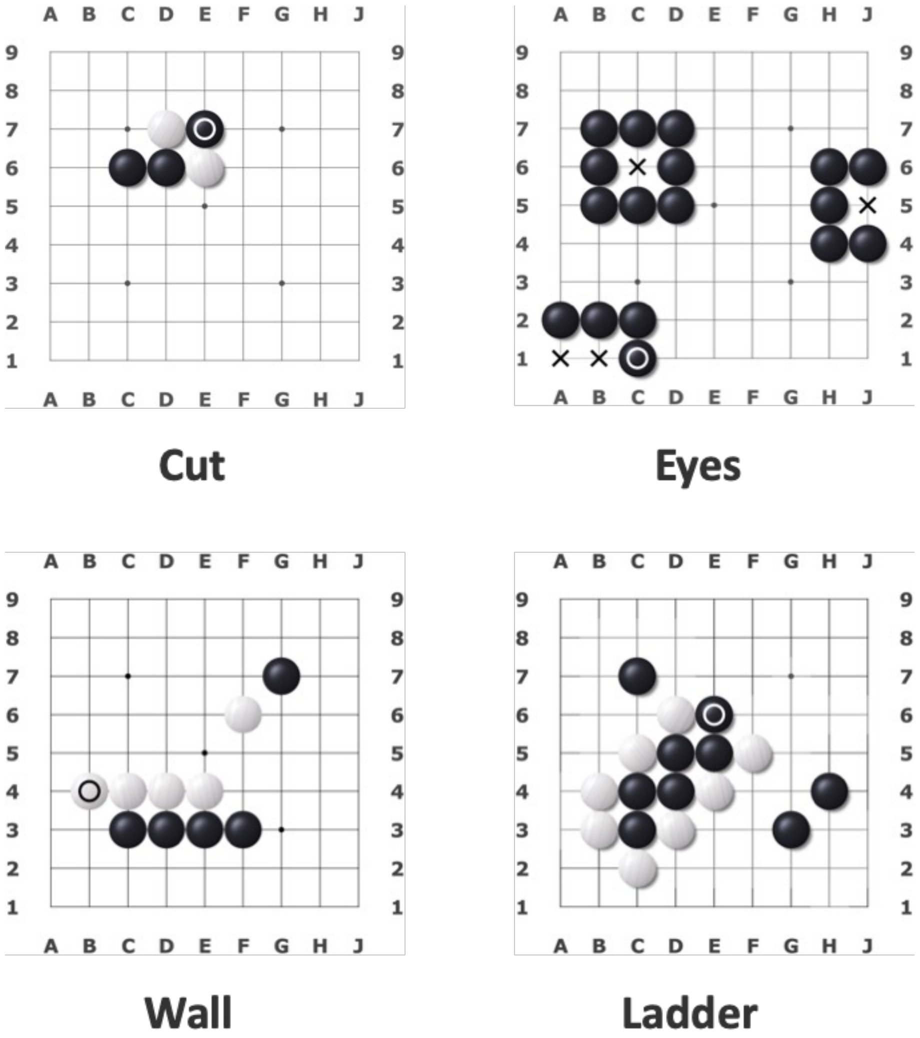
Model-Based RL: World Models

Model-Based RL: AlphaZero

- Learn both a policy and value network via self-play (reward of +1/-1 comes from end of game)
- Transition function is known: we can do explicit planning
- Use Monte Carlo tree search (MCTS) to choose actions based on the current value function

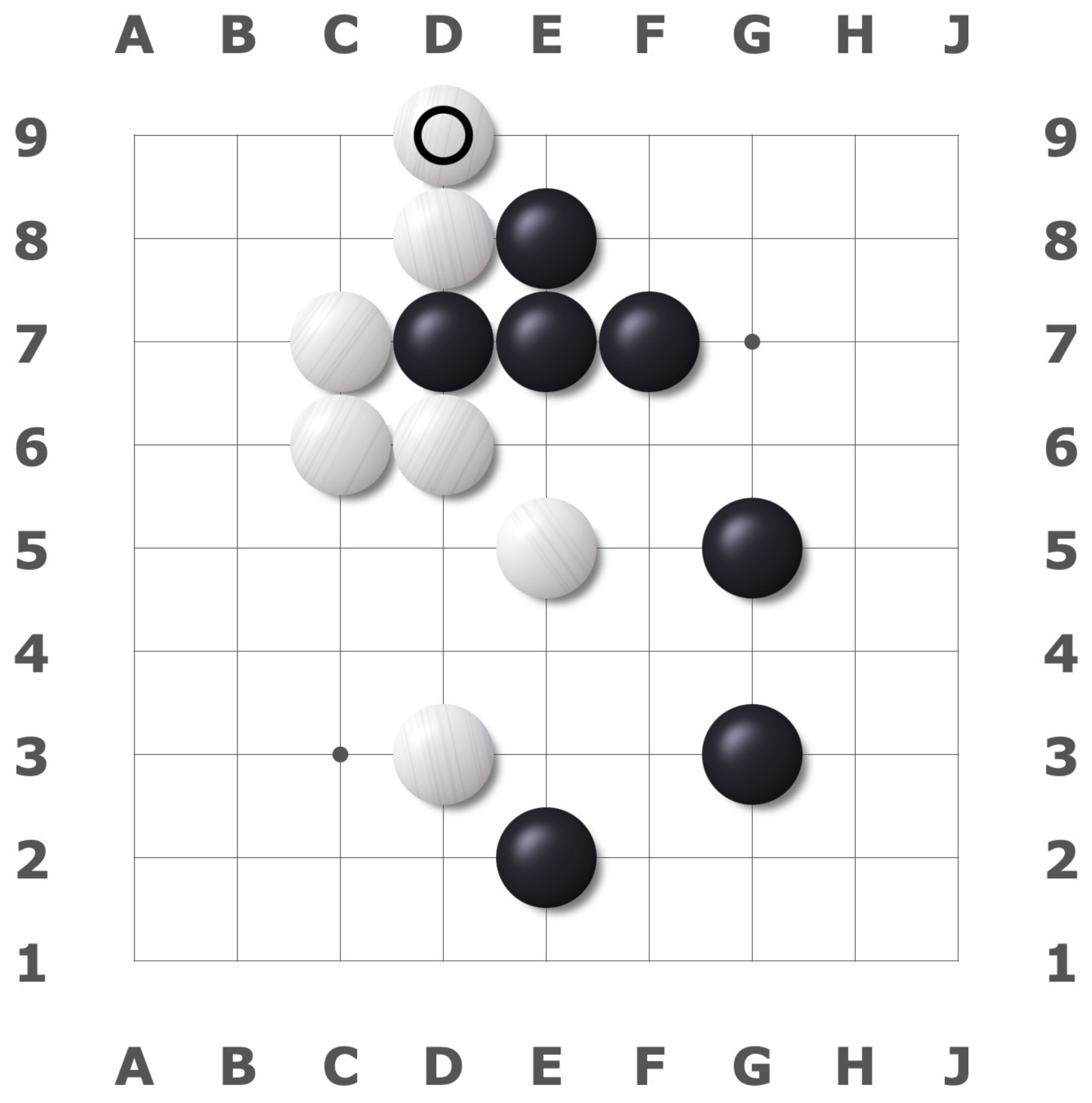


What does AlphaGo Zero Learn? (Tomlin, et al. 2022)



Concept	Definition
Eye	Surrounded empty space
Wall	Sequence of stones in a row
Ladder	Zig-zag capturing race
Pincer	Attack on a corner approach
Joseki	Fixed local sequence of moves
Sente	Initiative
Hane	Move that "reaches around"
Up	Toward the center of the board
Aji	Possibilities left in a position
Ko	Repeated capture sequence
Shape	Quality of stone arrangement
Gote	Loss of initiative
Moyo	Sphere of influence
Aji	Possibilities left in a position
Atari	Threat to capture

Extracting Concepts from Game States



Bad shape. If white wants to defend it should be solid at c8, leaving no weaknesses or sente moves for black.

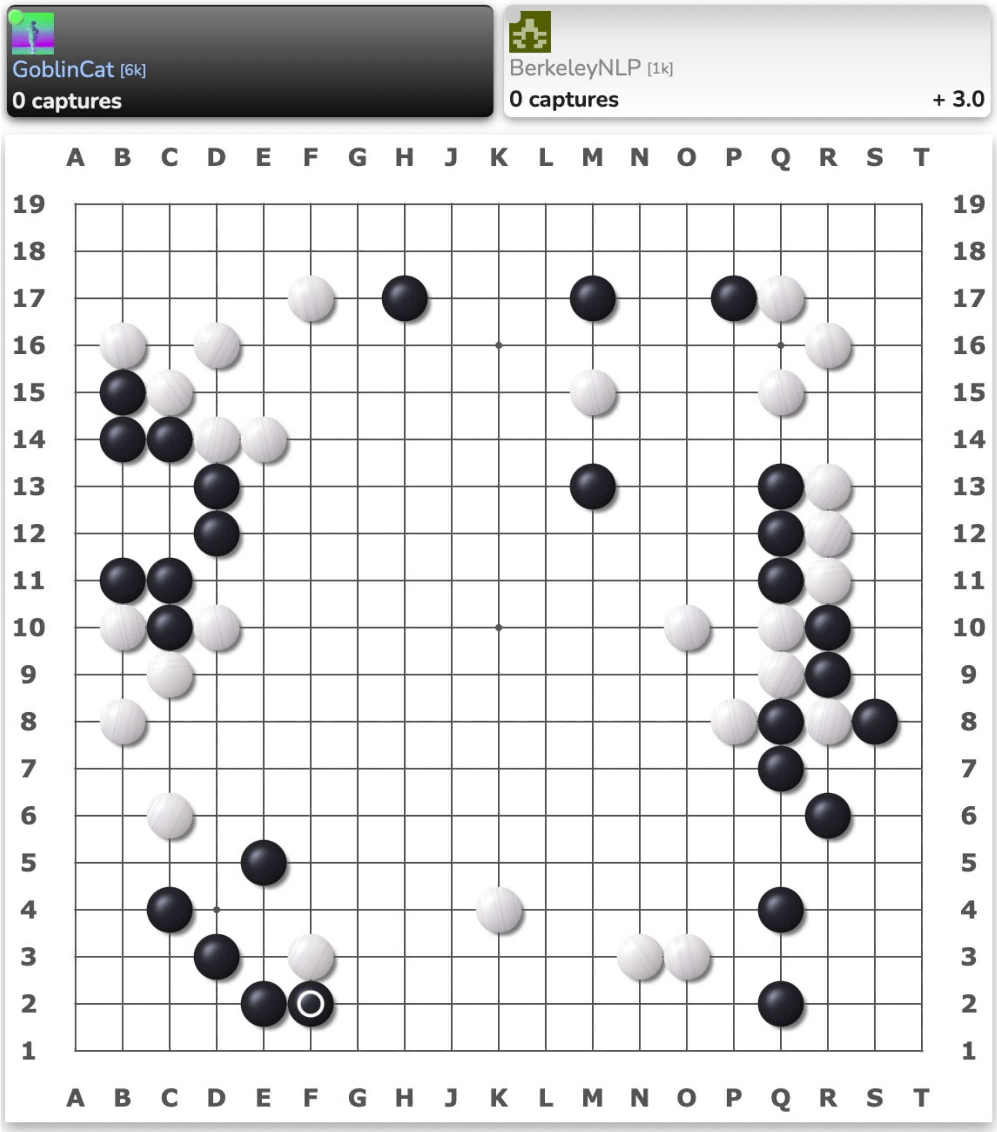
Dataset Statistics:

- 10K annotated games (19x19)
- Approximately 458K comments
- Additional data from unplayed variations (ignored in this work)

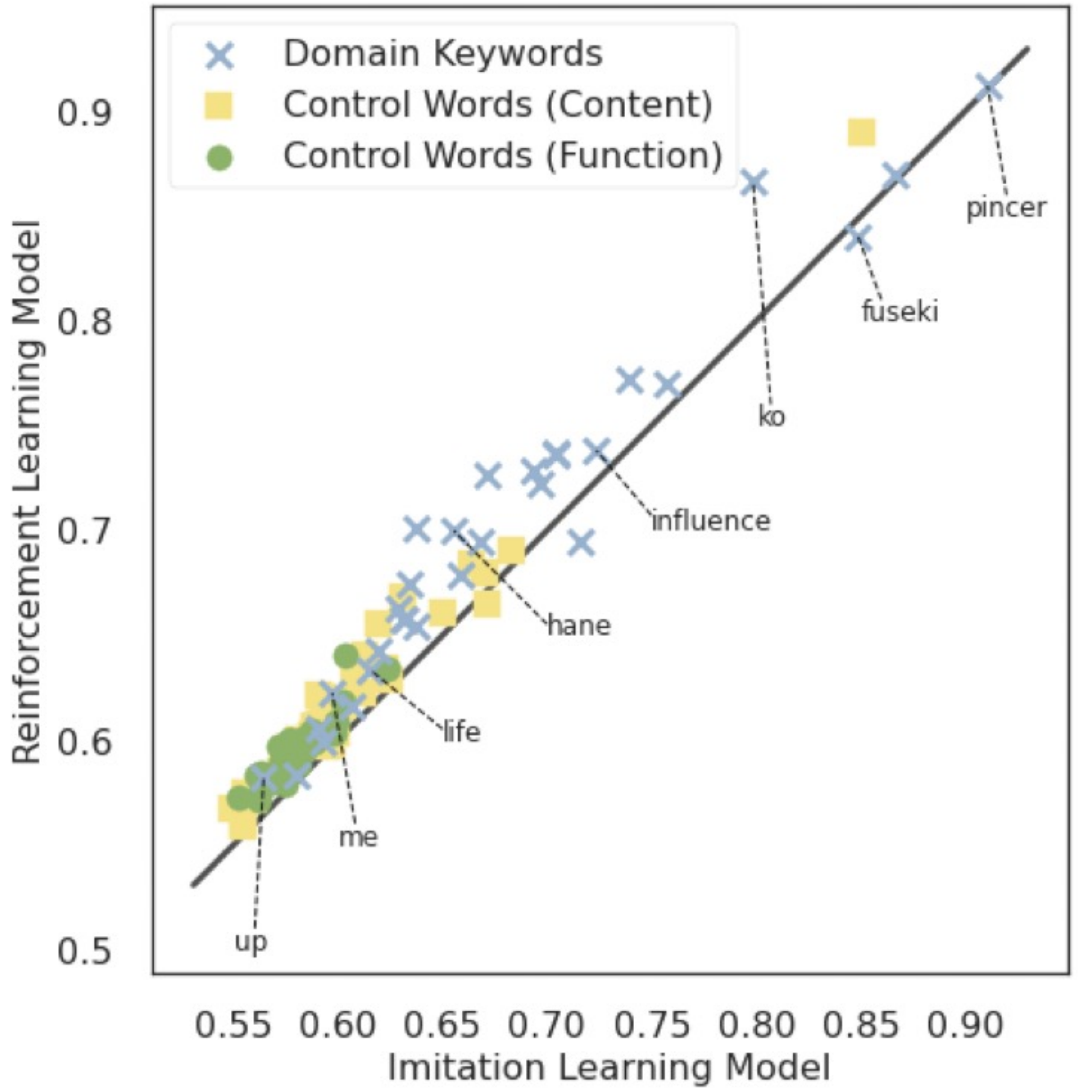
Game-Playing Agents

- **Agent #1: Imitation Learning**
  - Following CNN architecture from Clark and Storkey (2014)
  - Trained on 228K human games
  - Played against real humans on Online Go Server (OGS) and received a rating of 1K

- **Agent #2: Reinforcement Learning**
  - Pre-trained ELF OpenGo [Tian, et al. 2019]
  - Open-source equivalent of AlphaGo Zero [Silver, et al. 2017]
  - Better than all human players: ELO 5000+



Human-Level Concepts Encoded in Model



- Key finding: human-level concepts are predictable from the intermediate representations of both models
- Additionally: some concepts appear in early layers, and others in later layers (i.e., different levels of abstraction)
- **Long-term goal: how do we get human-interpretable explanations of models which exceed human capacity?**