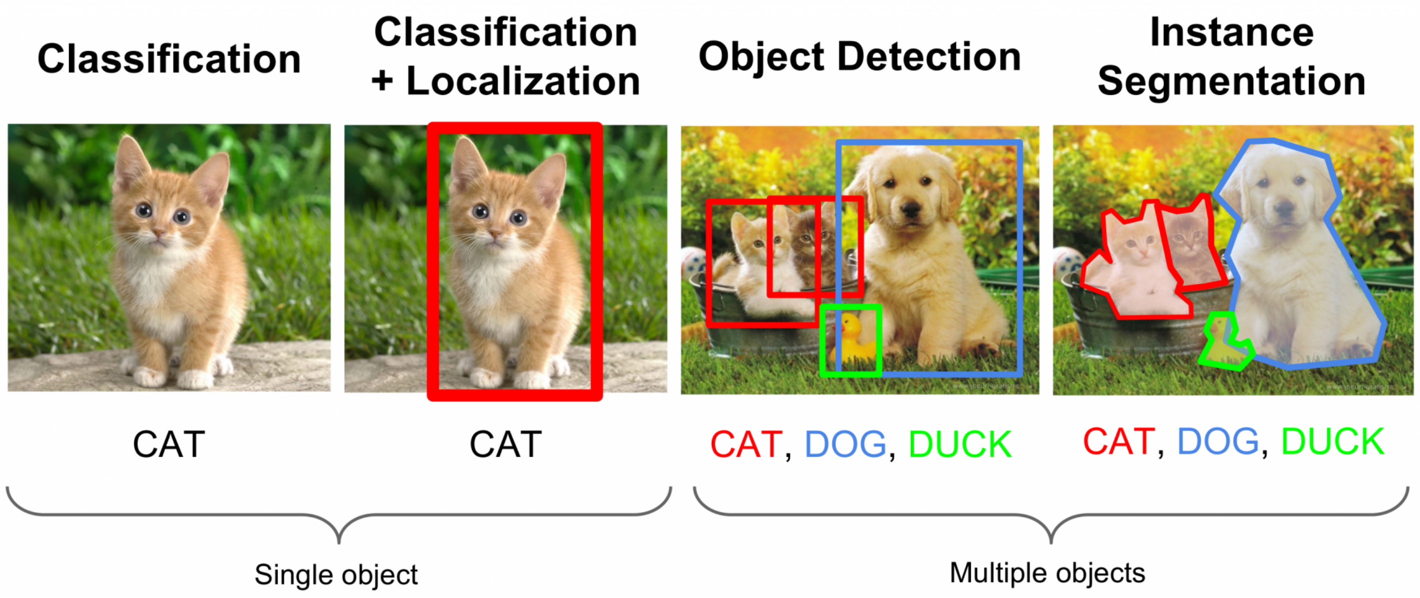
Intro to Reinforcement Learning

Gokul Swamy

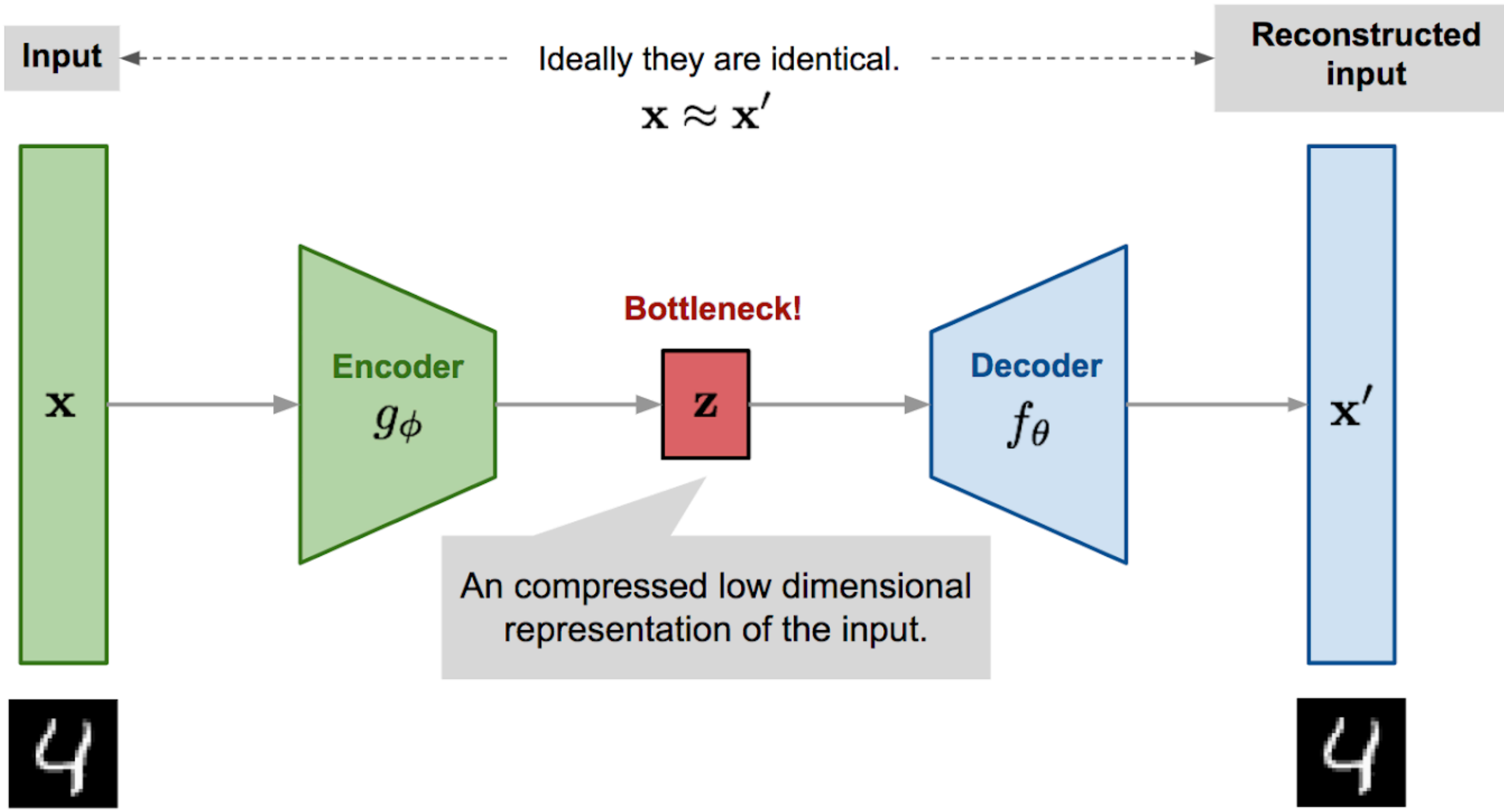
Supervised Learning

- Learn some sort of mapping from input to output that minimizes some notion of error
  - Learn how to take tests well

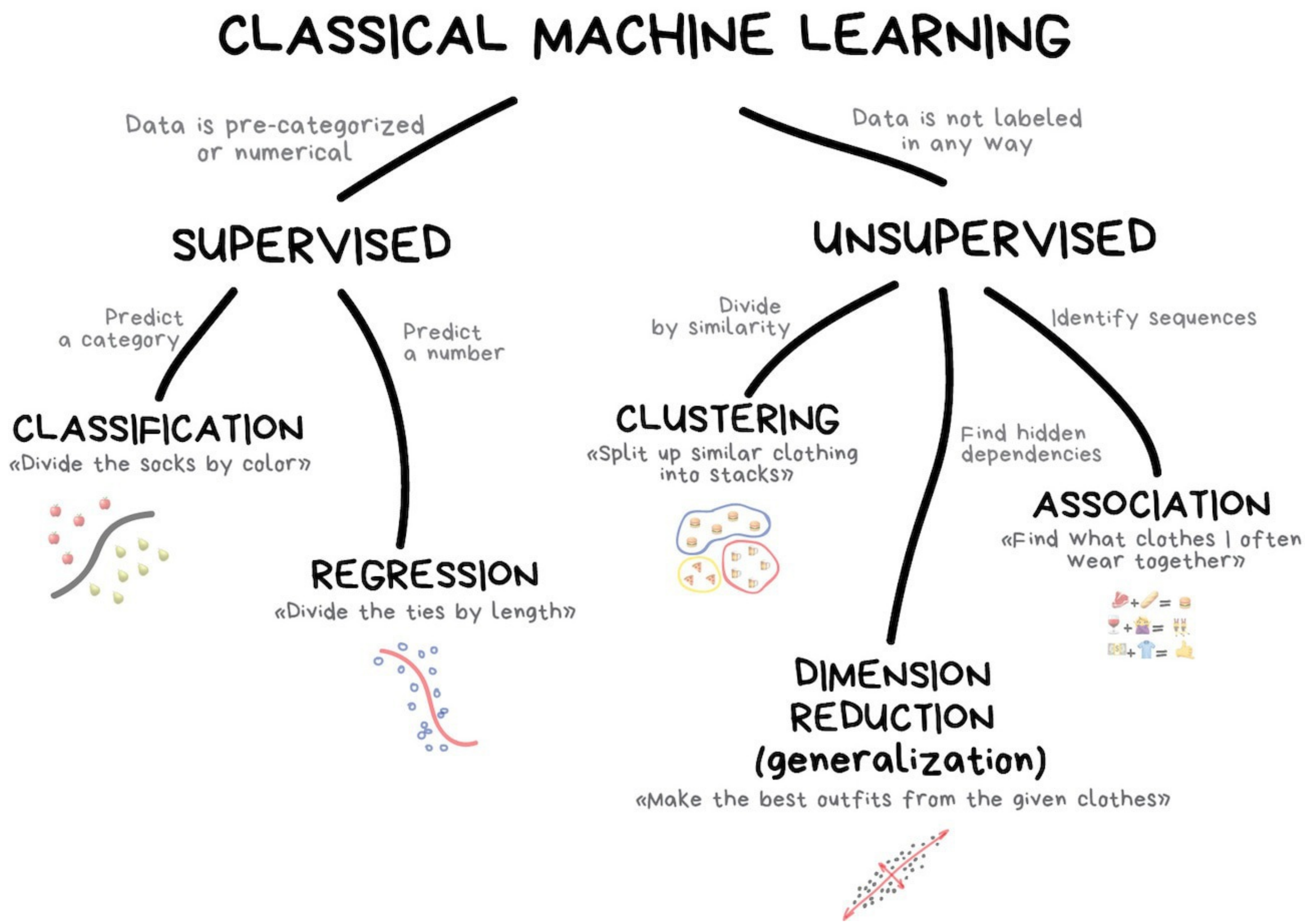


Unsupervised Learning

- Learn patterns to make downstream tasks easier
  - Clustering, auto-encoders, density models

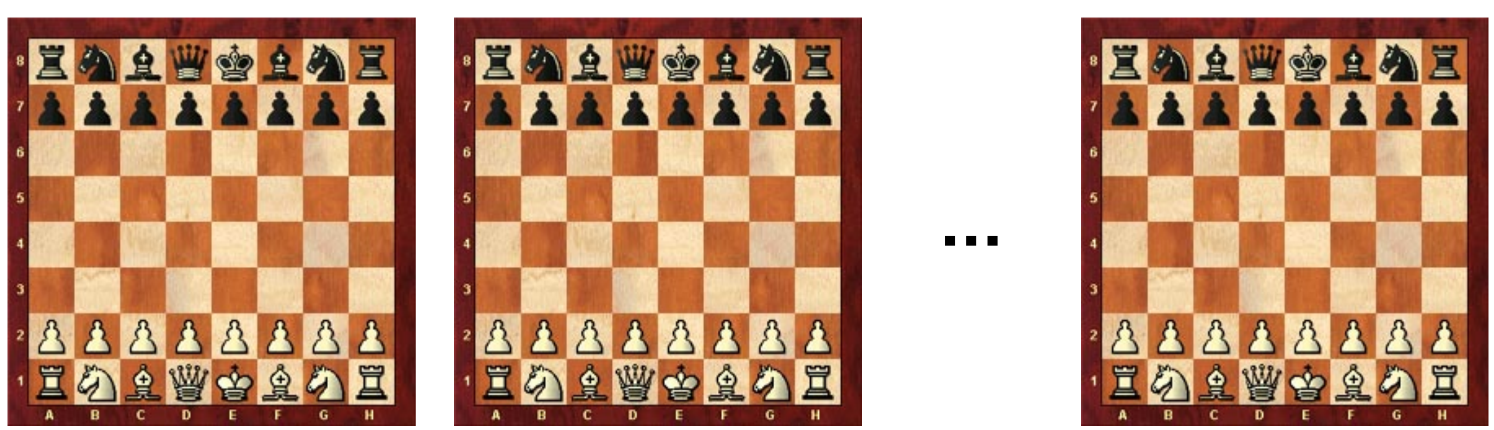


So far …


CLASSICAL MACHINE LEARNING
Data is pre-categorized or numerical
Data is not labeled in any way
SUPERVISED
UNSUPERVISED
Predict a category
Predict a number
Divide by similarity
Identify sequences
CLASSIFICATION
«Divide the socks by color»
REGRESSION
«Divide the ties by length»
CLUSTERING
«Split up similar clothing into stacks»
Find hidden dependencies
ASSOCIATION
«Find what clothes I often wear together»
DIMENSION REDUCTION (generalization)
«Make the best outfits from the given clothes»

What’s missing here?

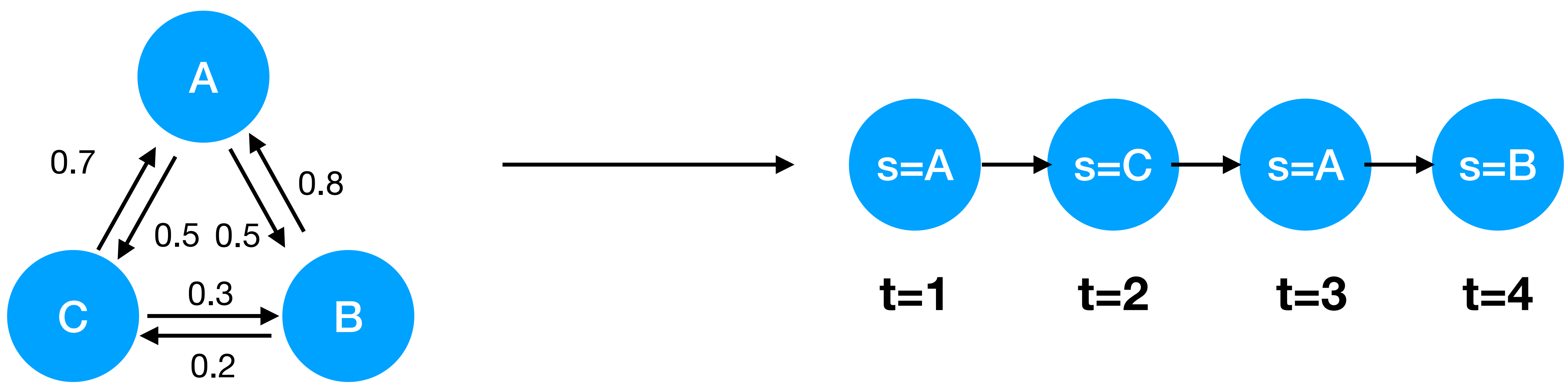
- Limited notion of one decision influencing the next input
    - Consider executing multi-step plans in chess
- Limited notions of dealing with uncertainty
    - How do you learn if there isn’t a precise label for what you’re doing?



T **T + 1** **T + N**

Introducing State: Markov Chains

- Model of a random process where at each timestep, the value of the random variable transitions
  - We're only going to consider discrete-time Markov Chains
- (First Order) Markov Property: The future is independent of the past conditioned on the present



Introducing Agency: Markov Decision Processes

Definitions

Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

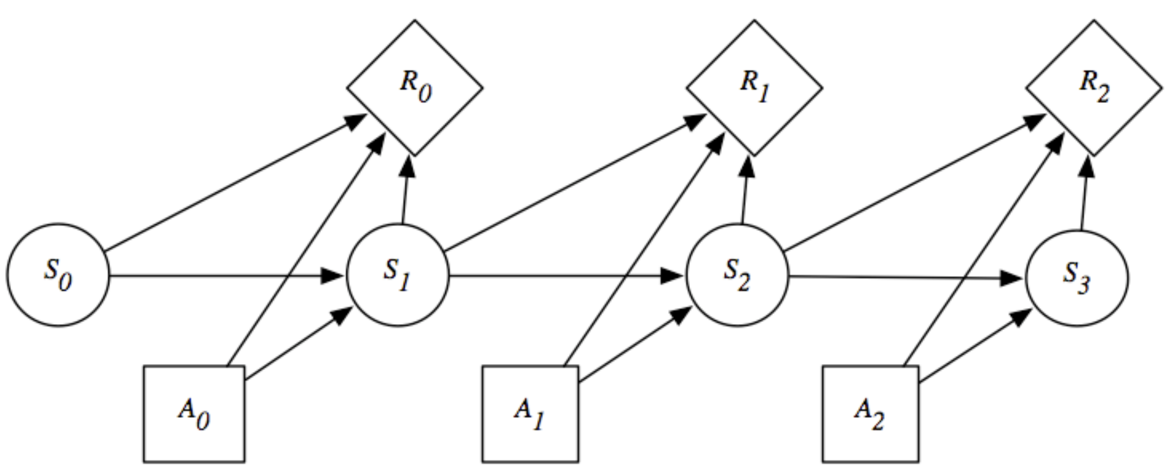
$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

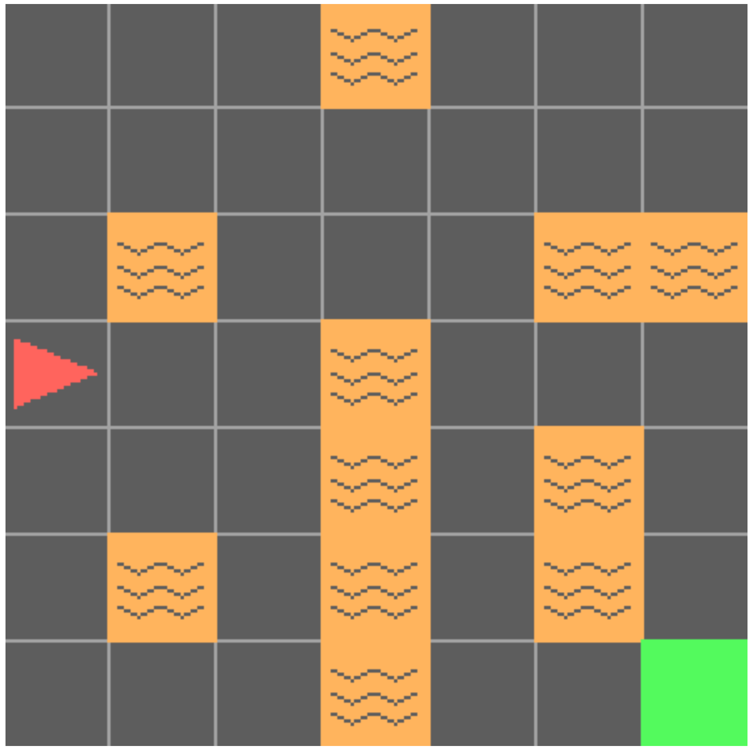
$\mathcal{T}$ – transition operator (now a tensor!) $T(s, a, s') = P(s' | s, a)$

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$r(s_t, a_t)$ – reward



Example:



Introducing Partial Observability: POMDPs

Definitions

partially observed Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

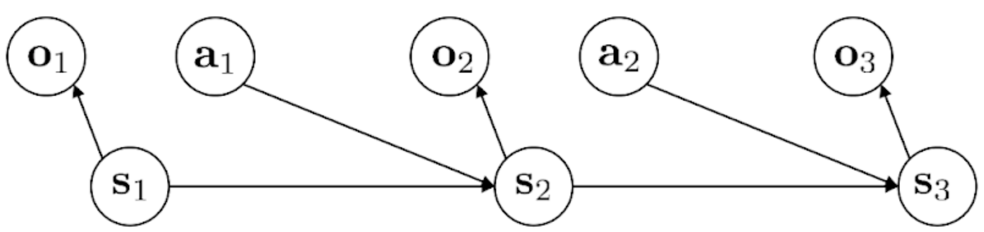
$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{O}$ – observation space observations $o \in \mathcal{O}$ (discrete or continuous)

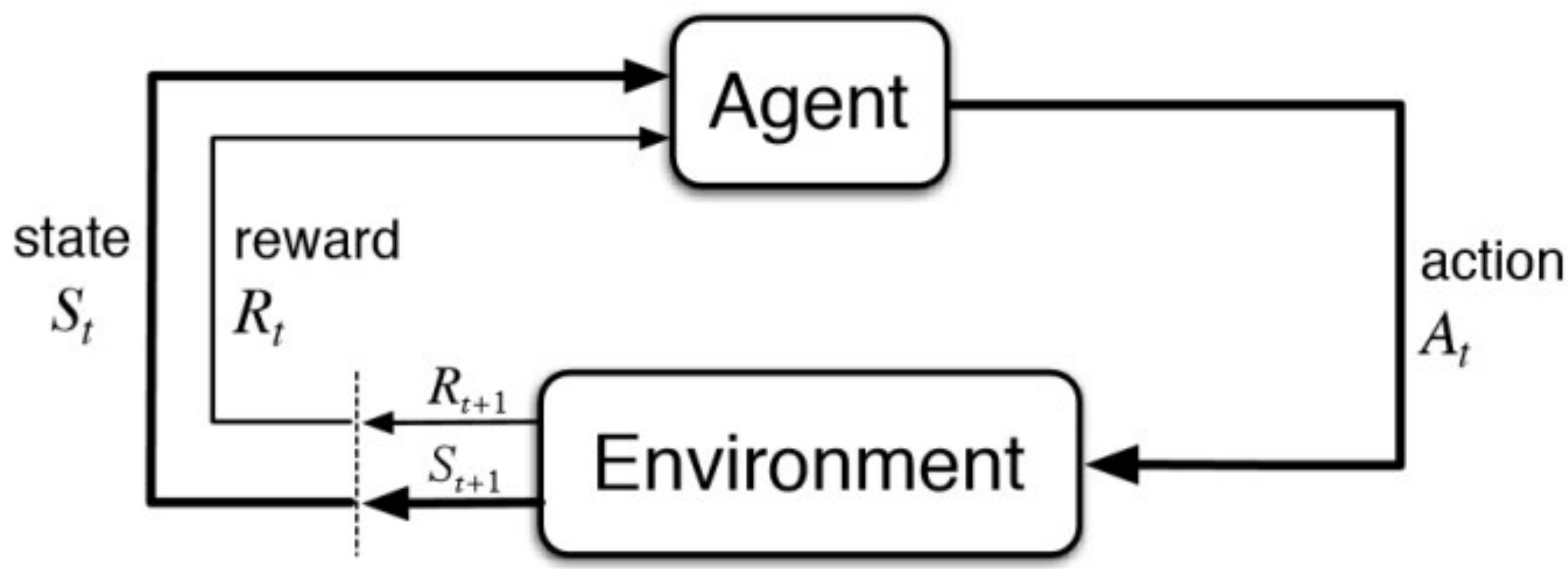
$\mathcal{T}$ – transition operator (like before)

$\mathcal{E}$ – emission probability $p(o_t|s_t)$

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$



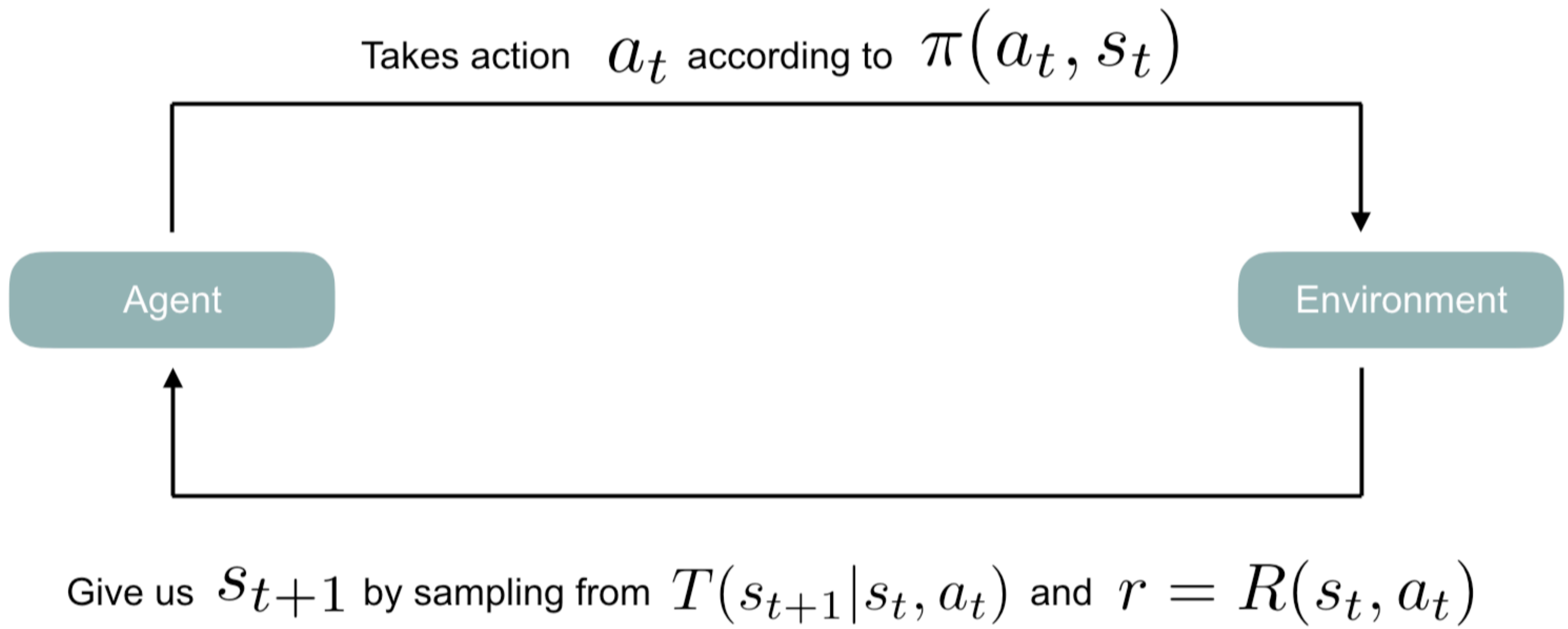
RL Framework



RL Definitions

- **Environment**: The world in which our problem is set up. The environment updates according to **dynamics**
- **State**: All the aspects of the environment at a particular time that are relevant to the problem we're trying to solve
- **Agent**: Can take actions to influence the state of the world
- **Policy**: How our agent decides to act given the state of the world. A distribution over actions conditioned on state.
- **Trajectory**: List of state-action tuples generated by our interaction with env.

RL Framework Formalized



The Reinforcement Learning Objective

Maximize expected utility!

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s}, \mathbf{a})}$$

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

Value and Q Functions

- Discounted sum of future rewards:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- The average of this defines the "value" of a state:

$$V^\pi(s) = \mathbb{E}_\pi \left[R_t | s_t = s \right]$$

- We can break this down even further to actions:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[R_t | s_t = s, a_t = a \right]$$

Combining Value and Q Functions

- Off Policy:

$$V^{\pi}(s) = \max_{a} Q(s,a)$$

- On Policy:

$$V^{\pi}(s) = E_{\pi}[Q(s,a)]$$

- Advantage Function:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

Known T/R + Discrete States: Policy Iteration

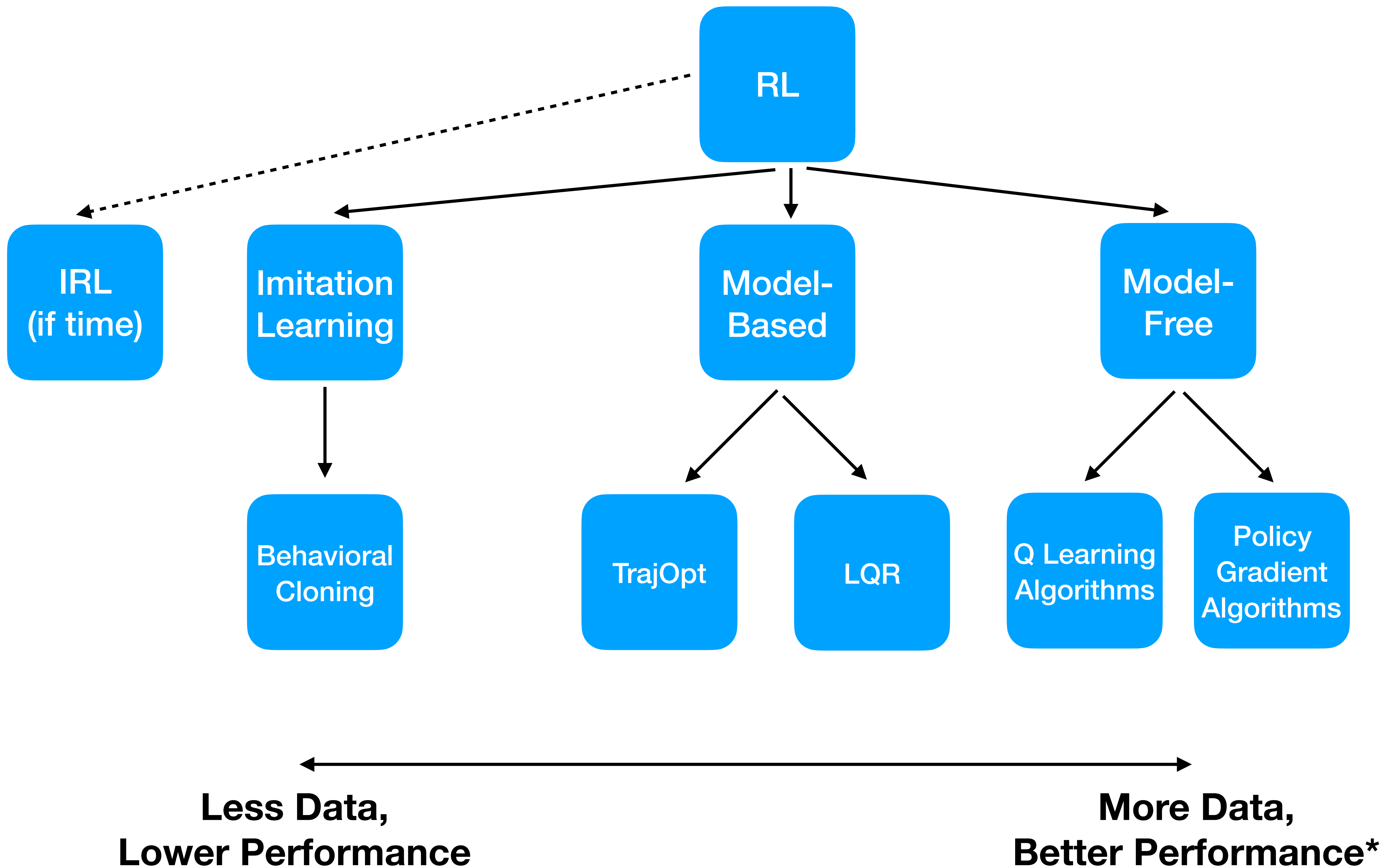
- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')\right]$$

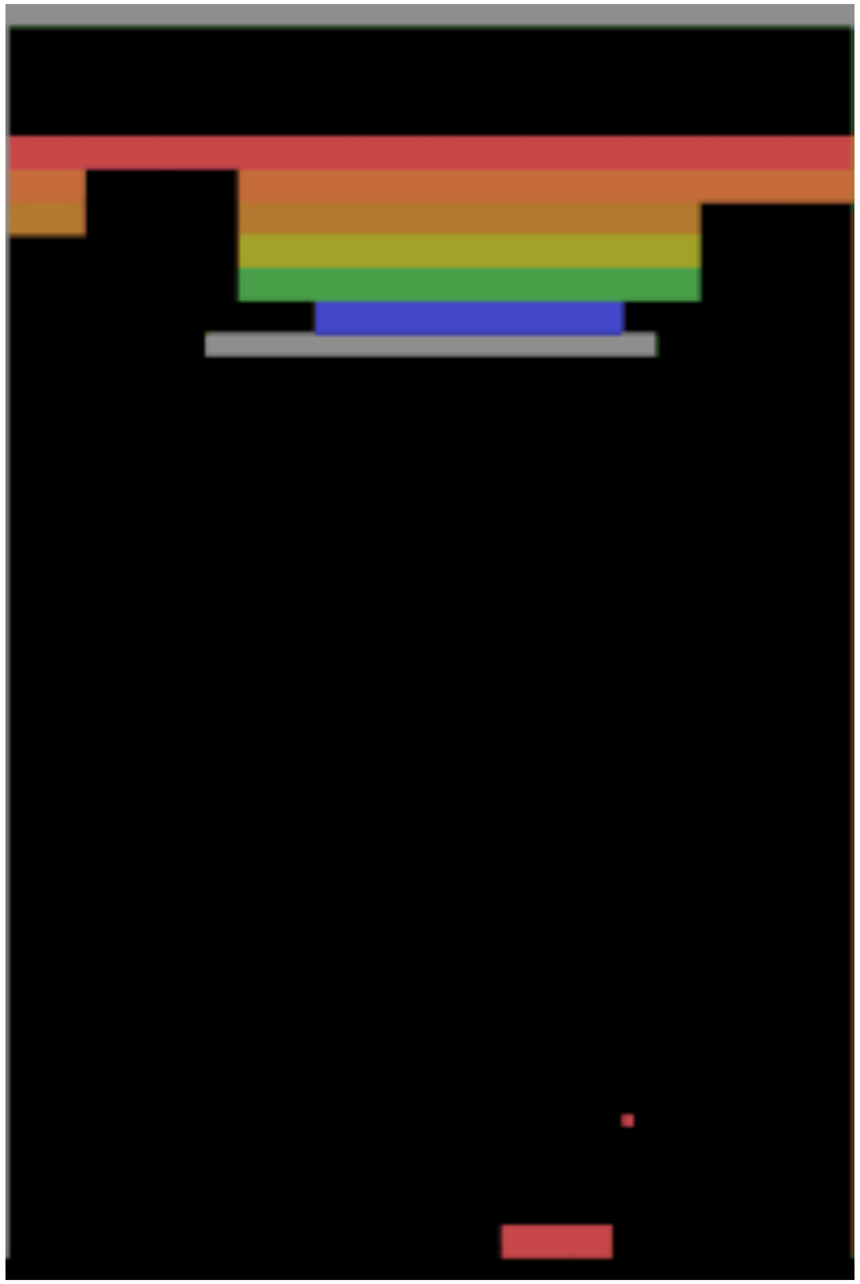
- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Taxonomy of RL Algorithms



Running Example: Breakout



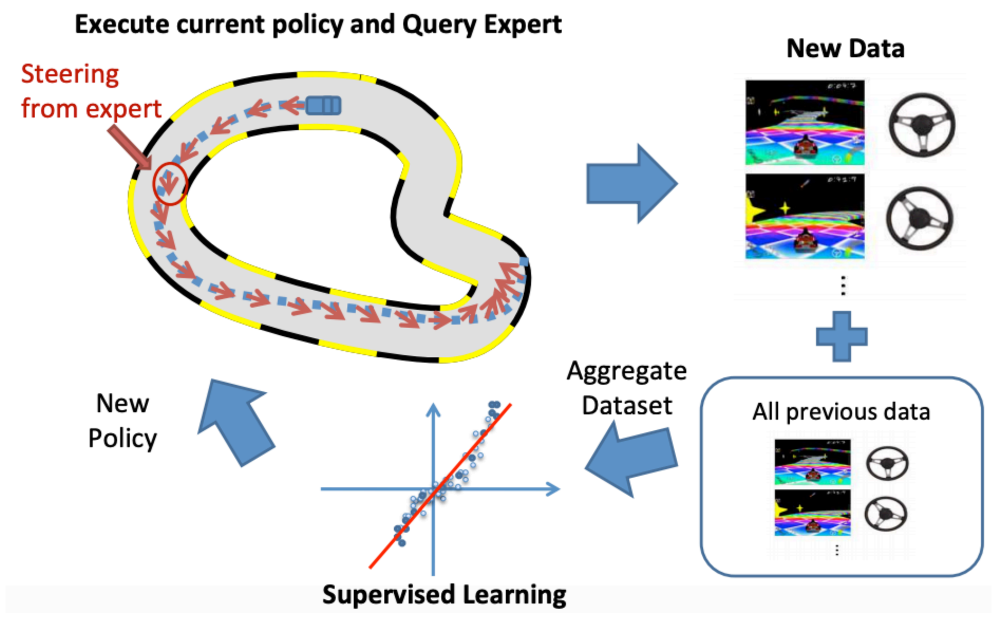
Imitation Learning: Behavioral Cloning

- Given demonstrations of a person performing a task, learn a function that maps from states to their actions
  - This is a completely supervised approach to learning a policy
  - Does not generalize well: imagine trying to drive a car

$$loss = \frac{1}{N} \sum_{i} (\pi(s_i) - a_i)^2$$

Imitation Learning: DAGGER

1. train $\pi_\theta(u_t|o_t)$ from human data $\mathcal{D}_{\pi^*} = \{o_1, u_1, ..., o_N, u_N\}$
2. run $\pi_\theta(u_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, ..., o_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions u_t
4. Aggregate: $\mathcal{D}_{\pi^*} \leftarrow \mathcal{D}_{\pi^*} \cup \mathcal{D}_\pi$
5. GOTO step 1.



(For more info, read https://www.ri.cmu.edu/pub_files/2015/3/InvitationToImitation_3_1415.pdf)

Model-Based RL (1)

- Learn a function that captures dynamics / rewards of system
- Then, use classical planning methods to optimize your reward function

model-based reinforcement learning version 1.0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute those actions and add the resulting data $\{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_j\}$ to $\mathcal{D}$

Why is f hard to learn?

Model-Based RL (2)

- What if you make an error while predicting what's going to happen?

  - Replan at every timestep (Model Predictive Control)

model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

every N steps

Planning: iLQR

- Approximate dynamics as linear function
- Approximate cost (negative reward) as quadratic function
- Then, there exists an easily computable optimal set of controls (actions)

$$
\begin{aligned}
x_{t+1} &= Ax_t + Bu_t, \ t \in \{0, 1, \ldots, N\} \\
x_0 &= x^{init}
\end{aligned}
$$

$$
J(U, x_0) = \sum_{\tau=0}^{N-1} \left(x_\tau^T Q x_\tau + u_\tau^T R u_\tau \right) + x_N^T Q_f x_N
$$

$$
\begin{aligned}
u_t^* &= -K_t z_t \\
P_t &= Q + K_t^T R K_t + (A - BK_t)^T P_{t+1} (A - BK_t), \ P_N = Q_f \\
K_t &= (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A.
\end{aligned}
$$

Planning: TrajOpt

- Turn planning into constrained optimization problem
  - Cost: Learned reward function
  - Constraints: Learned dynamics
- Use convex approximation of above to solve in real-time
  - Expand and shrink trust region based on how accurate approximation is

Planning: TrajOpt

1: **for** PenaltyIteration $= 1, 2, \ldots$ **do**
2:     **for** ConvexifyIteration $= 1, 2, \ldots$ **do**
3:         $\tilde{f}, \tilde{g}, \tilde{h} = \text{ConvexifyProblem}(f, g, h)$
4:         **for** TrustRegionIteration $= 1, 2, \ldots$ **do**
5:             $\mathbf{x} \leftarrow \arg\min_{\mathbf{x}} \tilde{f}(\mathbf{x}) + \mu \sum_{i=1}^{n_{ineq}} |\tilde{g}_i(\mathbf{x})|^+ + \mu \sum_{i=1}^{n_{eq}} |\tilde{h}_i(\mathbf{x})|$
                subject to trust region and linear constraints
6:             **if** TrueImprove / ModelImprove $> c$ **then**
7:                 $s \leftarrow \tau^+ * s$ ▷ Expand trust region
8:                 **break**
9:             **else**
10:                 $s \leftarrow \tau^- * s$ ▷ Shrink trust region
11:             **if** $s <$ xtol **then**
12:                 **goto** 15
13:         **if** converged according to tolerances xtol or ftol **then**
14:             **break**
15:     **if** constraints satisfied to tolerance ctol **then**
16:         **break**
17:     **else**
18:         $\mu \leftarrow k * \mu$

Model-Free RL

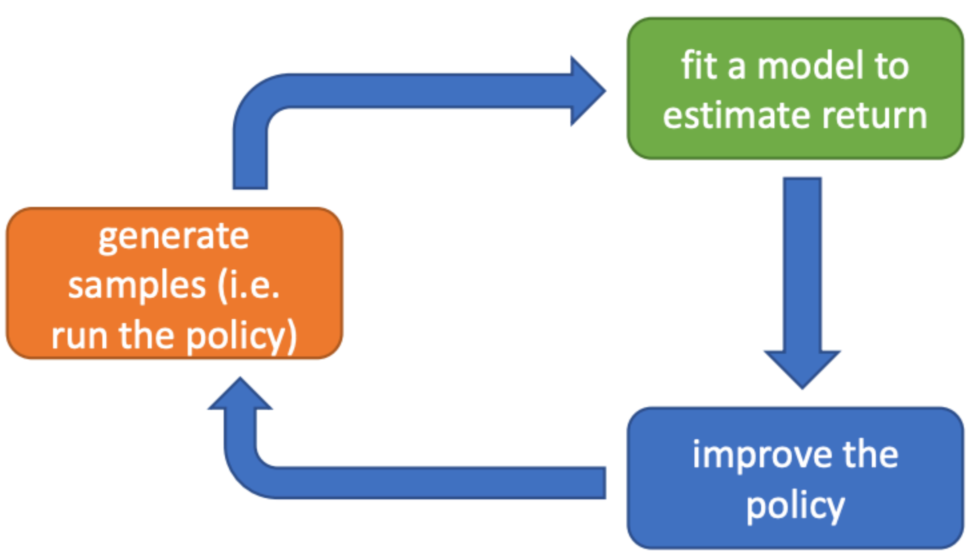
- Instead of learning the dynamics, why don't we learn just the policy
  - In some sense, this is all we're really after
  - More compact representation of correct action to take
  - Sometimes, a full dynamics model is unnecessary and harder to learn (extraneous entries in state)
  - More info: https://arxiv.org/pdf/1805.00909.pdf

Q Learning

- With the correct Q function, policy is just argmax over actions
  - Off-Policy!

Tabular Q-Learning

$$Q_\phi(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}',\mathbf{a}')$$



$$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a})$$

DQNish*

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

***= hacks like target network to work**

Soft Actor-Critic (Extra)

Soft Policy Iteration

1. Soft policy evaluation:

Fix policy, apply soft Bellman backup until converges:

$$Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \mathbb{E}_{\mathbf{s}' \sim p_{\mathbf{s}},\ \mathbf{a}' \sim \pi} \left[Q(\mathbf{s}', \mathbf{a}') - \log \pi(\mathbf{a}' | \mathbf{s}') \right]$$

This converges to Q^π.

2. Soft policy improvement:

Update the policy through information projection:

$$\pi_{\text{new}} = \arg\min_{\pi'} \mathrm{D}_{\mathrm{KL}} \left(\pi'(\cdot | \mathbf{s}) \,\middle\|\, \frac{1}{Z} \exp Q^{\pi_{\text{old}}}(\mathbf{s}, \cdot) \right)$$

For the new policy, we have $Q^{\pi^{\text{new}}} \geq Q^{\pi^{\text{old}}}$.

3. Repeat until convergence

Soft Actor-Critic

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.* ICML, 2018.

1. Take one stochastic gradient step to minimize soft Bellman residual
2. Take one stochastic gradient step to minimize the KL divergence
3. Execute one action in the environment and repeat

Policy Gradient (1)

- What if we try to directly optimize RL objective through gradient descent instead of learning a Q function?

  - Why might this be a good idea?

    - First, lets abbreviate our objective as

$$J(\theta) = E_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} R(s_t, a_t)]$$

    - Then, we can substitute and differentiate to get us

$$\pi_\theta(\tau)\nabla_\theta \log(\pi_\theta(\tau)) = \nabla_\theta \pi_\theta(\tau)$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta}[\nabla_\theta \log(\pi_\theta(\tau)) \sum_{t=0}^{T} R(s_t, a_t)]$$

Policy Gradient (2)

$$\begin{aligned}
\nabla_\theta \log(\pi_\theta(\tau)) &= \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right] \\
&= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \\
&= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}
\end{aligned}$$

Policy Gradient (3)

- This gives us a gradient as follows:

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta}[(\sum_{t=1}^{T} \nabla_\theta \log(\pi_\theta(a_t|s_t))(\sum_{t=1}^{T} R(s_t, a_t))]$$

- We can use a sample average as an unbiased estimator

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N}((\sum_{t=1}^{T} \nabla_\theta \log(\pi_\theta(a_t|s_t)))(\sum_{t=1}^{T} R(s_t, a_t)))$$

Policy Gradient (4)

- Putting it all together:
  - 1) Collect $\{\tau_1, \tau_2, ..., \tau_N\}$ by running policy in simulator
  - 2) Compute gradient according to formula

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}((\sum_{t=1}^{T}\nabla_\theta \log(\pi_\theta(a_t|s_t)))(\sum_{t=1}^{T}R(s_t, a_t)))$$

  - 3) Update parameters through gradient ascent

$$\theta' = \theta + \alpha\nabla_\theta J(\theta)$$

Policy Gradient (5)

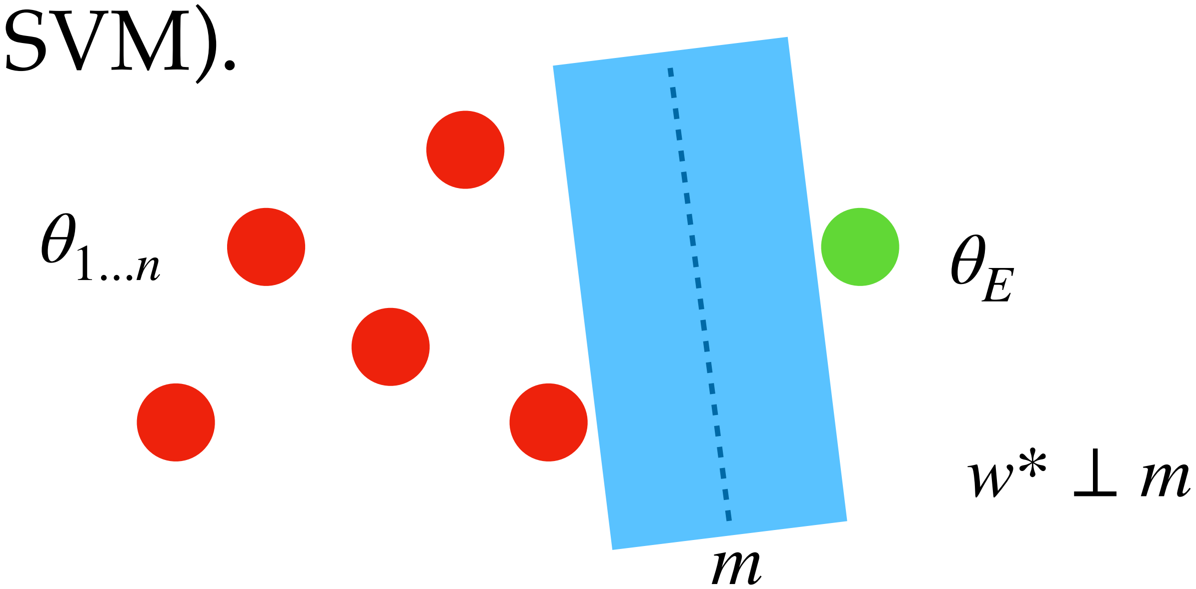
- This will not work very well. There are lots of add-ons to get this to work:
  - Causality
  - Advantage Functions
  - Actor-Critic
  - Surrogate Objectives + Clipping
- Still incredibly sensitive to wacky things like network initialization with all of these and more

What algorithm should I use?

- Real world:
  - Most problems: Iterative LQR
  - Very complex problem: model-based RL - learn a dynamics model (maybe a deep network) and then use TrajOpt
- Simulated:
  - Data is very easy to collect: PPO
  - Data is harder to collect: Soft Actor-Critic

Inverse Reinforcement Learning

- Reinforcement Learning: find actions that maximize reward
- Inverse Reinforcement Learning: find reward function that would have made actions taken optimal
  - Standard Recipe: write $R(s) = w^T \theta(s_E, a_E)$ with fixed features and find w s.t. the reward is greater than for any other set of state-action pairs (effectively max-margin SVM).



Fixing Traditional IRL

- Will match expert feature counts at convergence
- Requires demonstrator to be optimal at each time step
- Unfortunately, this is really hard to ensure in practice, especially if your data comes from people
- More reasonable assumption from cognitive science: Boltzmann Rationality:
  - $\mathbb{P}((s_i, a_i)|R) = \frac{1}{Z_i}\exp\{\alpha Q^*(s_i, a_i, R)\}$
  - Here, expert is exponentially more likely to take an action with a higher Q value rather than having all probability mass on a single action
  - Has the same expected feature counts as idealized distribution so fits into the above framework

MaxEnt Inverse Reinforcement Learning

- Remember that a Q value is a sum of rewards, each of which follows the linear form we had before
- We apply Bayesian Inference to recover reward function:

$$\mathbb{P}(\tau|R) = \prod_{i=1}^{n} \mathbb{P}((s_i, a_i)|R)$$

$$\mathbb{P}(\tau|R) = \frac{1}{Z}\exp\{\alpha \sum_{i=1}^{n} Q^*(s_i, a_i, R)\}$$

$$\mathbb{P}(R|\tau) = \frac{\mathbb{P}(\tau|R)\mathbb{P}(R)}{\mathbb{P}(\tau)} = \frac{1}{Z}\exp\{\alpha \sum_{i=1}^{n} Q^*(s_i, a_i, R)\}\mathbb{P}(R)$$

- We can then maximize this objective w.r.t w via gradients

Popular RL Packages

```
pip install tensorflow

pip install gym

git clone https://github.com/openai/baselines.git

git clone https://github.com/rail-berkeley/
softlearning.git

cd baselines

pip install -e .

cd ../softlearning

pip install -e .
```

Resources

- Pretty much everything you see here was shamelessly copied from one of:
  - Anca's CS 188 Slides
    - Including the slide template
  - Pieter's CS 287 Slides
  - Sergey's CS 285 (CS 294-112) Slides
  - Claire's EE 221a Notes
- Best place to learn more online: https://spinningup.openai.com/en/latest/
- Best classes to learn more in real life: Above Professors' grad classes