Reinforcement Learning (Basics)

Roshni Iyer

April 2020

Slides Adapted from UC Berkeley: CS 285 Course (Deep Reinforcement Learning, Decision Making, and Control)
Instructor: Sergey Levine, Fall 2019
http://rail.eecs.berkeley.edu/deeprlcourse/

Content

- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

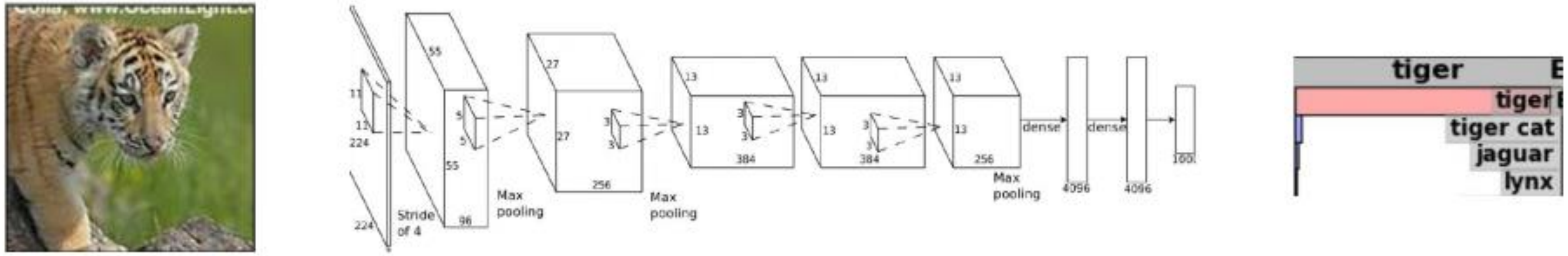
Content

- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

Terminology

- Unstructured environment (real-world):
  - Contains incomplete or irrelevant information for decision making
  - Not an ideal environment for the robot
- Deep Learning:
  - Enables for pattern-learning of large-scale data
  - Handles unstructured environments

Deep Learning Applications



EX: Robot traversing jungle

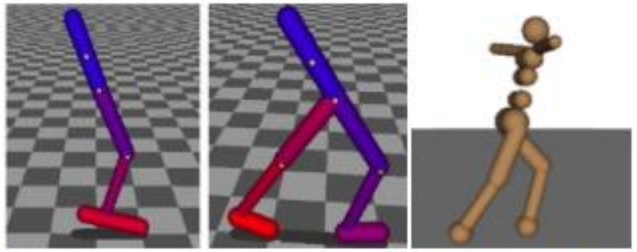
- Effectively handles raw sensory inputs (images, speech etc.)
- Tasks:
  - Image recognition
  - Machine translation
  - Speech recognition (QA)

Content

- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

RL provides a formalism for *behavior*

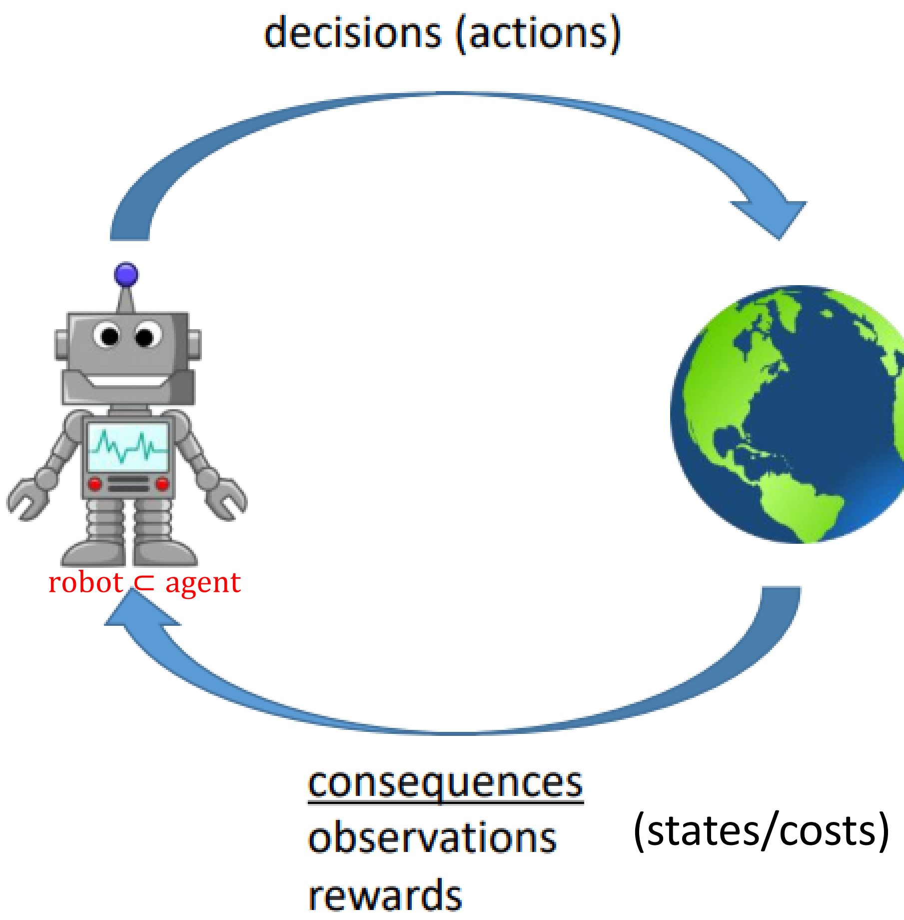
- Formalism for behavior to process data and <u>learn a mathematical model</u> for a decision-making problem
- Has the capacity to beat even the best human players



Schulman et al. '14 & '15

RL Operation

- Goal: Maximize reward/minimize cost
- Any AI problem = RL problem
  - 0 (1) in predicting label correctly (incorrectly) during training



Observations vs. States

- Observation:
  - subset of state
  - can contain aliasing (observations that appear the same but are in different states)
- State: describes everything that goes on in the world

$\mathbf{o}_t$ – observation

$\mathbf{s}_t$ – state

captures cheetah's presence

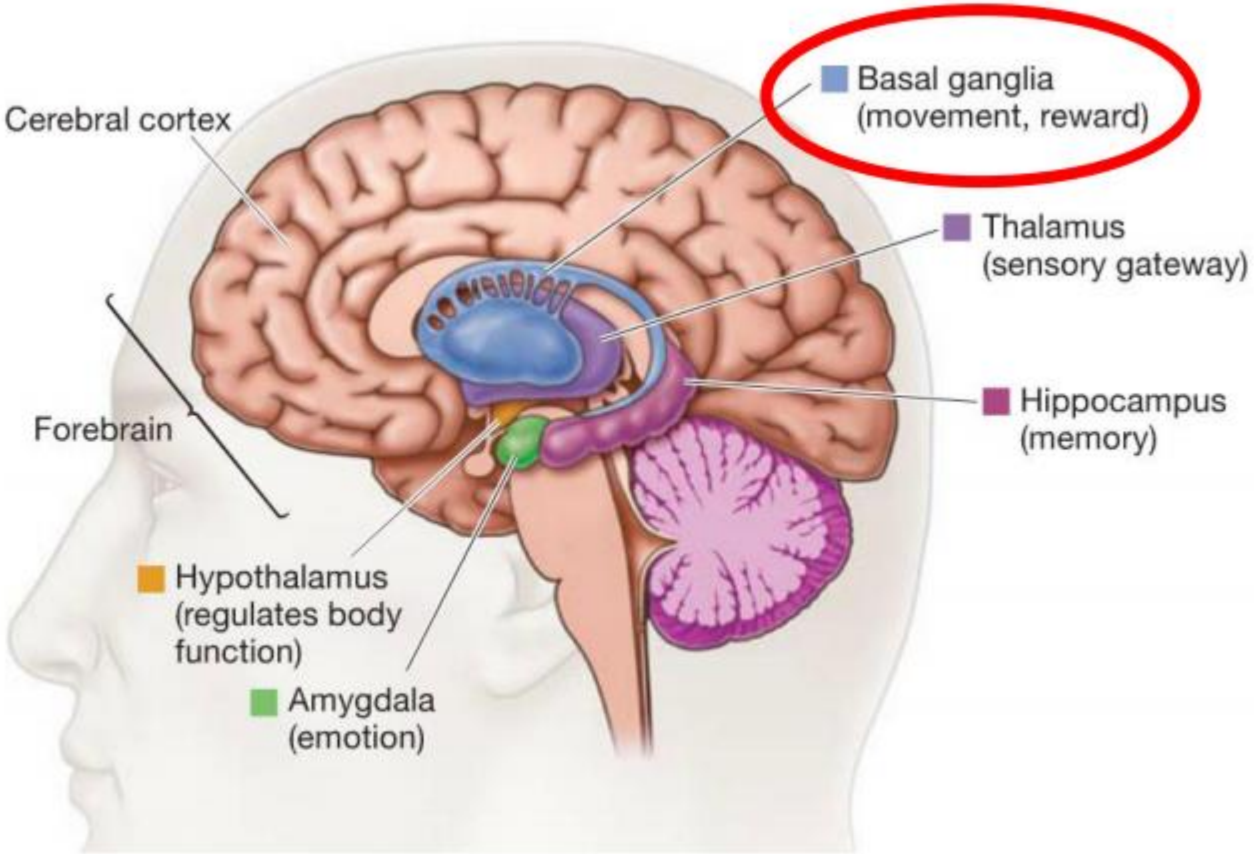
Content

- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

RL's Connection to the Human/Animal Brain

Trick → Treat → neuron fires
[eventually]
Trick → neuron fires

- Percepts that anticipate reward become associated with similar firing patterns as the reward itself



Other Forms of Reward Learning

- Inverse Reinforcement learning: Attempts to extract the reward function from the observed behavior of an agent
- Transfer Learning: Transferring knowledge between domains
- Meta-learning: Learning to learn

(jammed door → call maintenance; sink leak → ?)

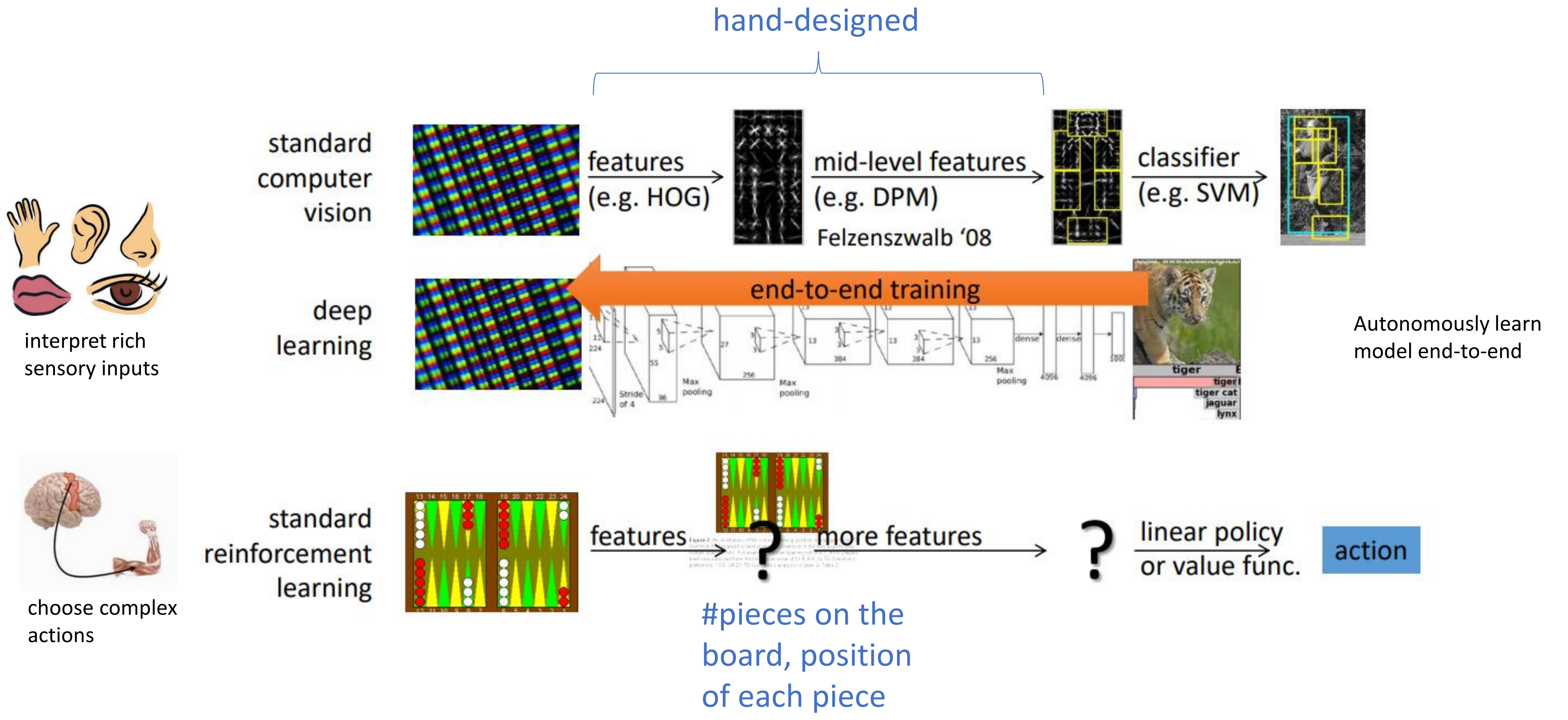
- Learning to predict and using prediction to act

  - Learning reward instead of policy (hunting in morning vs. night)
  - Cheetah hunting rabbits
  - Predicting direction of gazelle when hunting

Other Forms of Supervision

- Learning from demonstrations
  - Directly copying observed behavior
  - Inferring rewards from observed behavior (inverse RL)
- Learning from observing the world
  - Learning to predict
  - Unsupervised learning
- Learning from other tasks
  - Transfer learning
  - Meta-learning

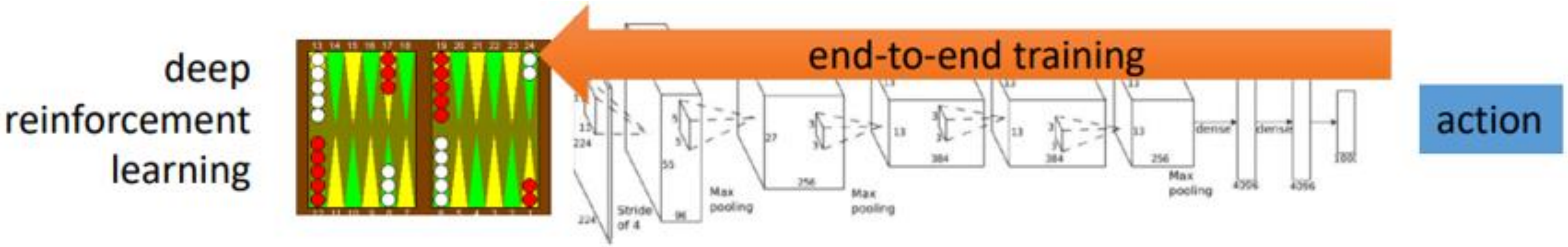
Deep Learning, Reinforcement Learning



Content

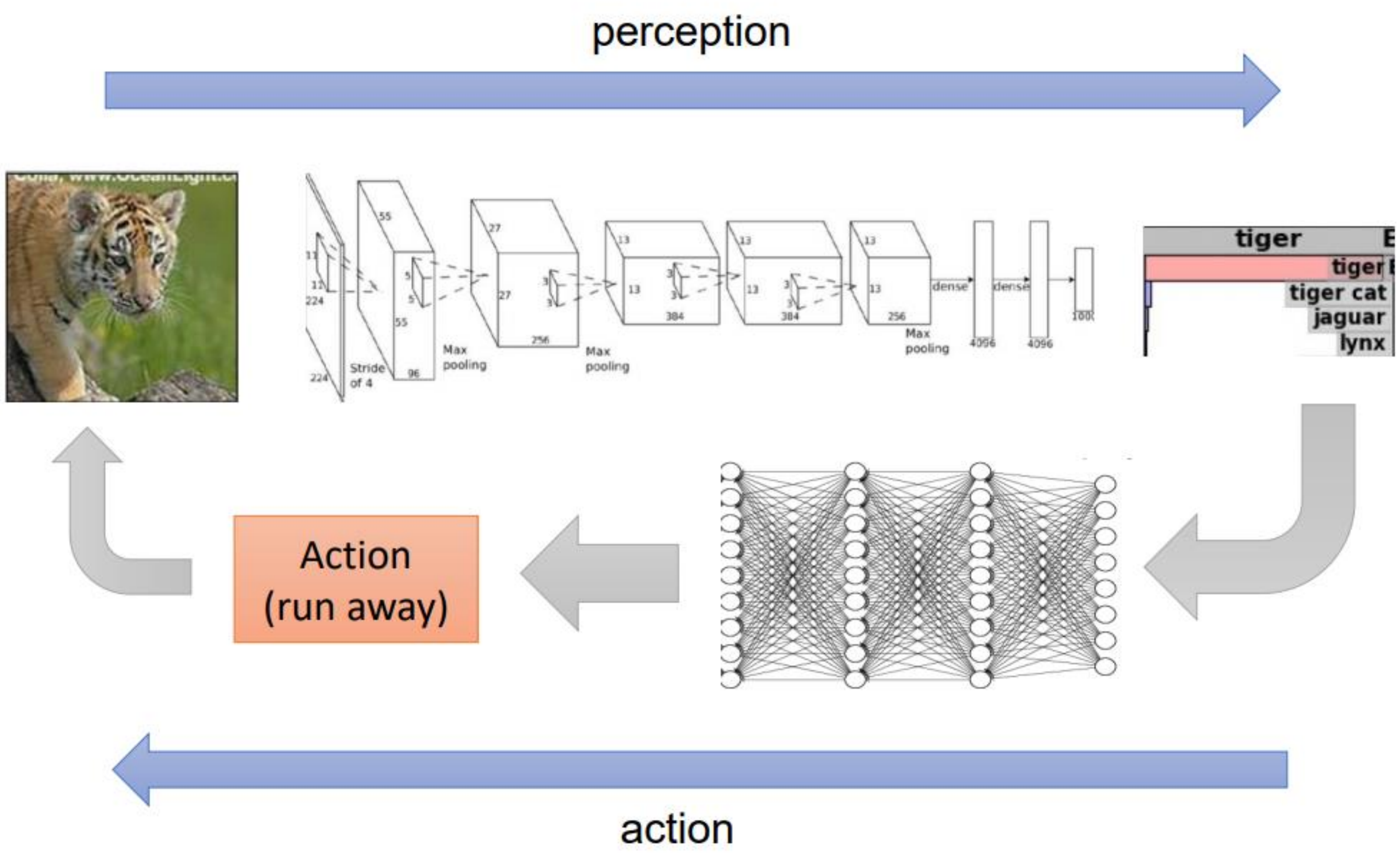
- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

A Single Algorithm: Deep Reinforcement Learning



can process complex sensory input + can choose complex actions

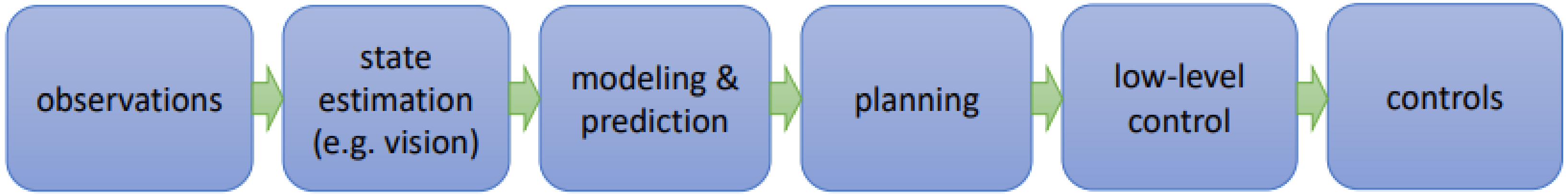
Deep RL Example



Deep RL Motivation

Executes the action determined, in the environment

Robotic control pipeline

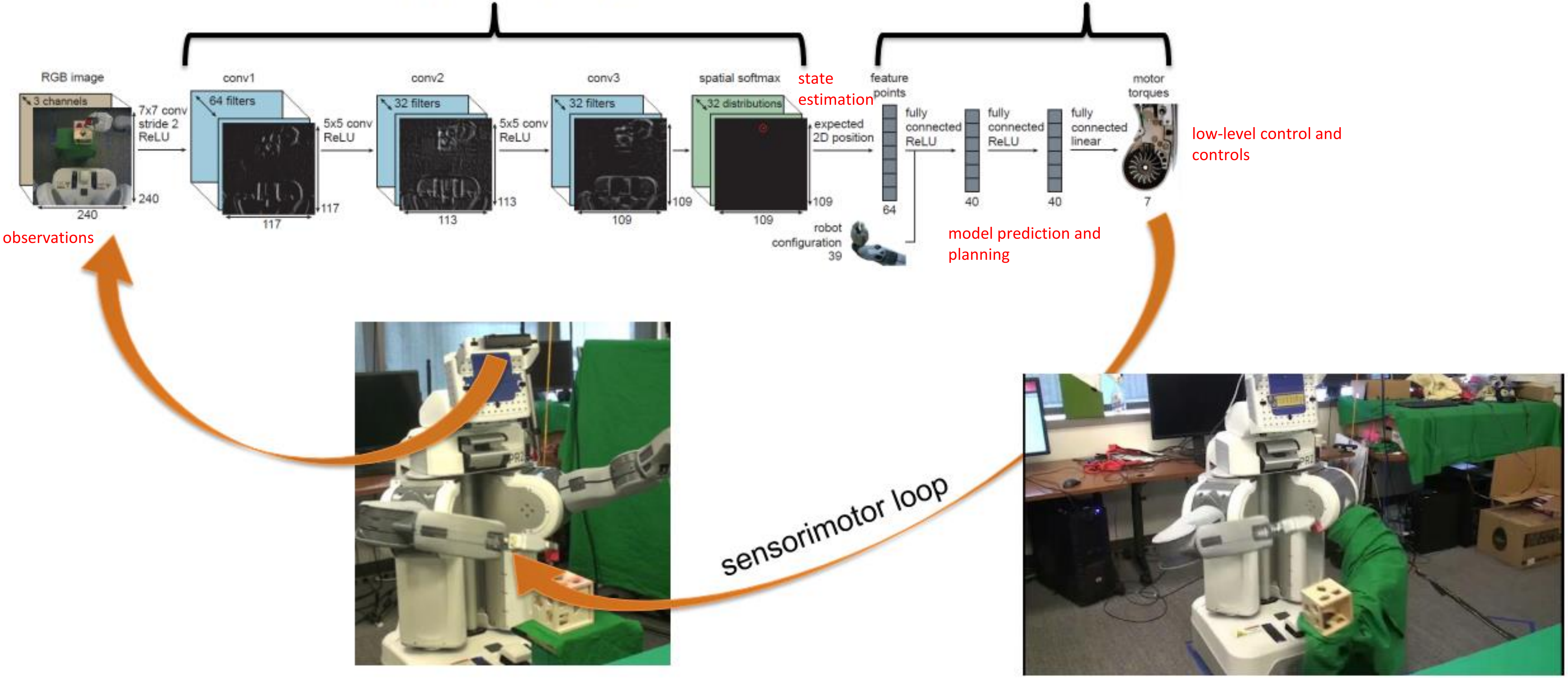


Cons of only using RL:

- Every phase in the pipeline can have some failure
- Failures compound to bigger errors if the entire component is not trained jointly **(state estimation is wrong → consequences of decisions are wrong)**

tiny, highly specialized "visual cortex"

tiny, highly specialized "motor cortex"



Deep RL Achievements

- Acquire high degree of proficiency in domains governed by simple, known rules
- Learn simple skills with raw sensory inputs, given enough experience
- Learn from imitating enough human provided expert behavior

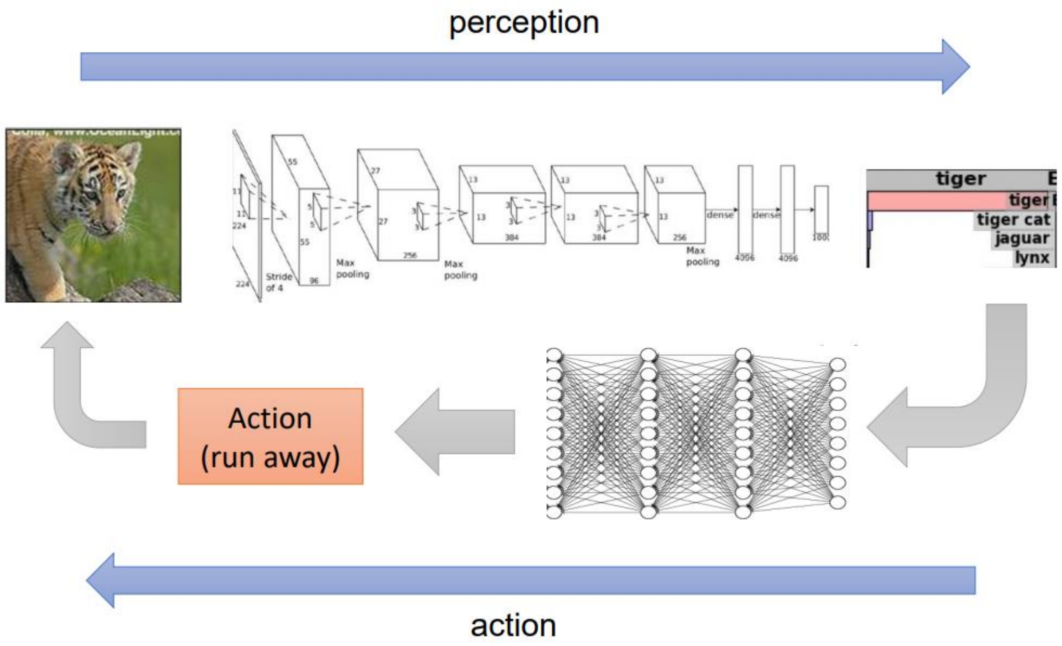
Deep RL Challenges

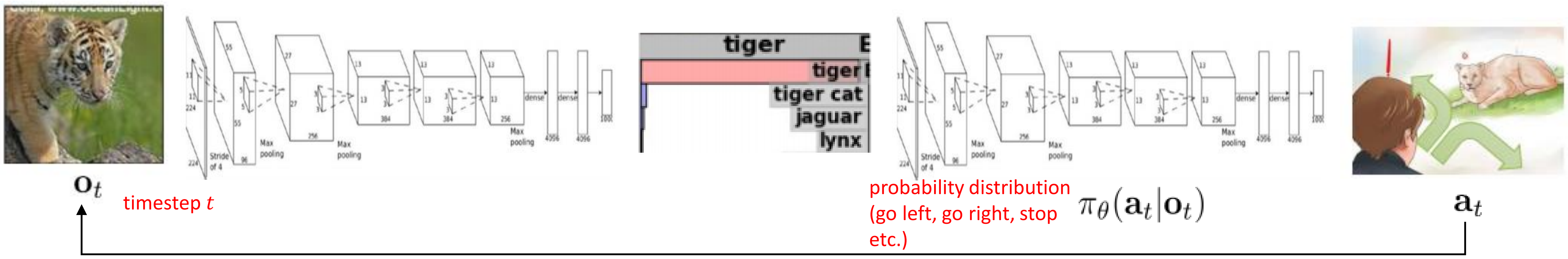
- Humans can learn incredibly quickly
  - Deep RL methods are usually slow
- Humans can reuse past knowledge
  - Transfer learning & meta-learning in deep RL is an open problem
- Not clear what the reward function should be
- Not clear what the role of prediction should be

Content

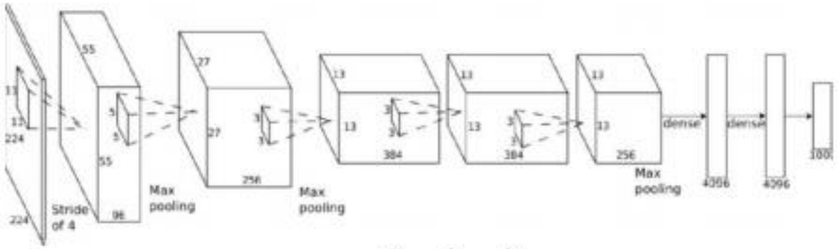
- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

Terminology + Notation





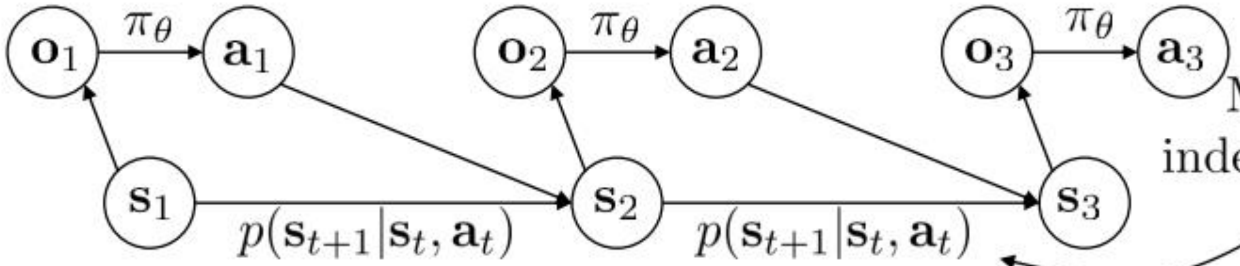
Terminology + Notation (cont.)



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

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

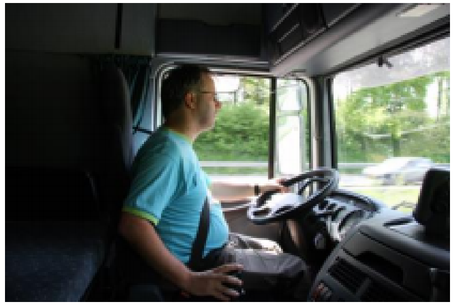
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)



Markov property
independent of $\mathbf{s}_{t-1}$

Only states satisfy
Markov property

Imitation Learning (specif. Behavioral Cloning)



$\mathbf{o}_t$
$\mathbf{a}_t$

training data

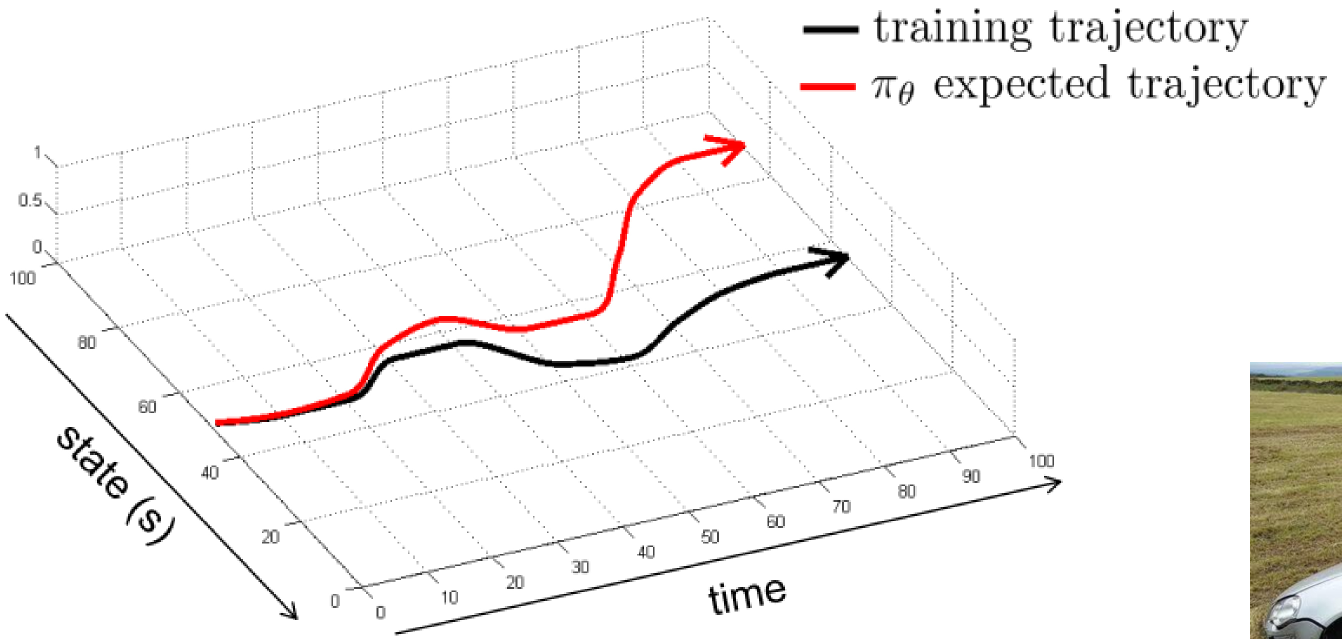
supervised learning

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

Imitation learning: supervised learning of good decision making (e.g., human expert)

Images: Bojarski et al. ‘16, NVIDIA

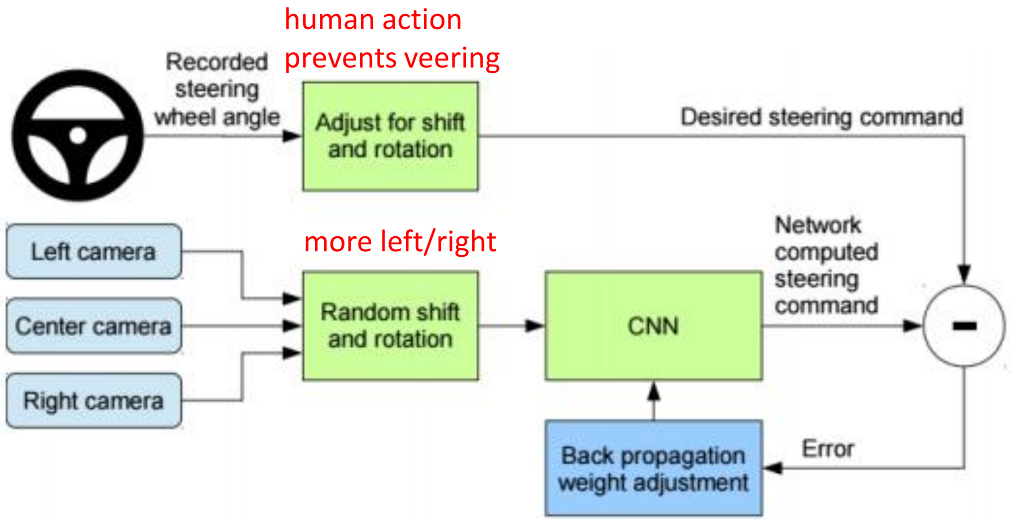
Does it work in general?

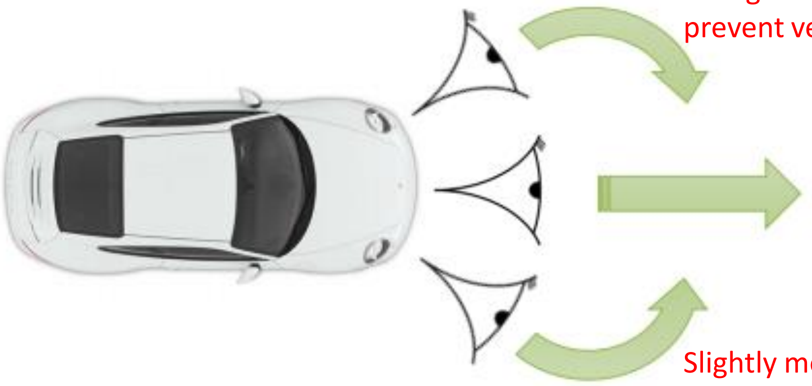


(No) – small mistakes compound to bigger mistakes due to unseen states (different situations) during test time

Why did that work?

- More leftward images were associated with more rightward steering actions in training data (likewise for more rightward images)
- Car learns a policy to prevent veering off the road
- **Alternative:** learn trajectories of more stable systems during training





Slightly more leftward facing camera to prevent veering to left

Slightly more rightward facing camera to prevent veering to right

Bojarski et al. '16, NVIDIA

Content

- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

How to make behavioral cloning work more often?

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

make the underlying distribution the same

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

DAgger: Dataset Aggregation

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

Ross et al. '11

Problems with the DAgger Algorithm

DAgger: Dataset Aggregation

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. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

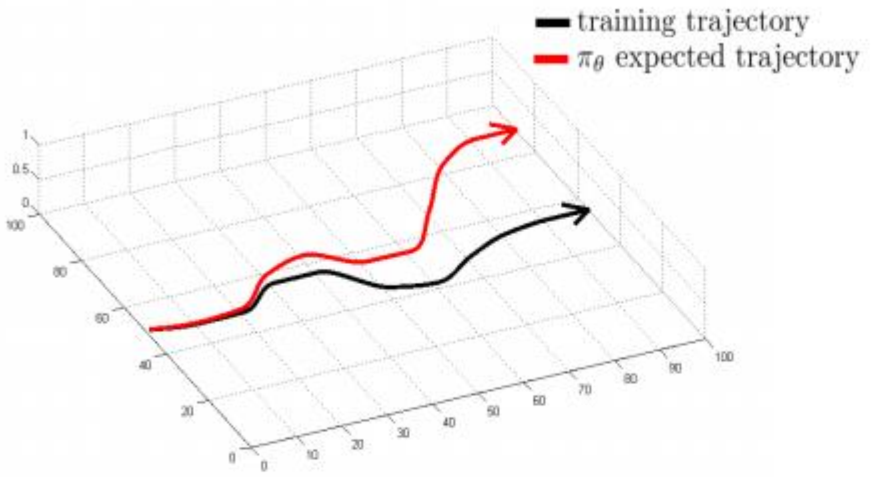
Ross et al. '11

Problems with the DAgger Algorithm

- Asking humans to provide action labels per iteration is unnatural
- Manual action labeling is prone to error
  - Labeling actions at discrete timesteps is not how actions (e.g., driving) are performed and omits the context of situation
  - **Non-Markovian Behavior:** Humans are inconsistent with labeling (different actions for same scenario – changing lanes)
  - **Multi-modal Behavior** (given assumed unimodal behavior)

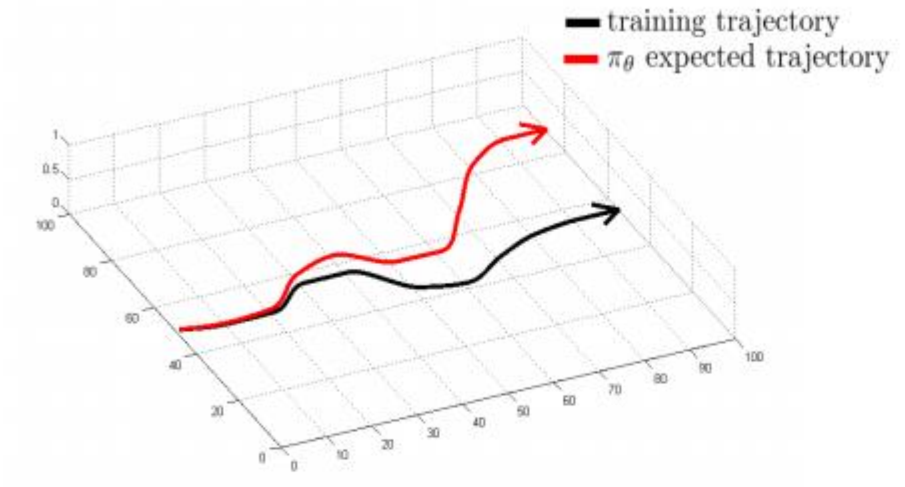
Can we make it work without more data?

- DAgger addresses the problem of distributional “drift”
- What if our model is so good that it doesn’t drift?
  - Need to use the whole history of data (contextual action – deer encounters)
  - Need to mimic expert behavior very accurately but don’t overfit

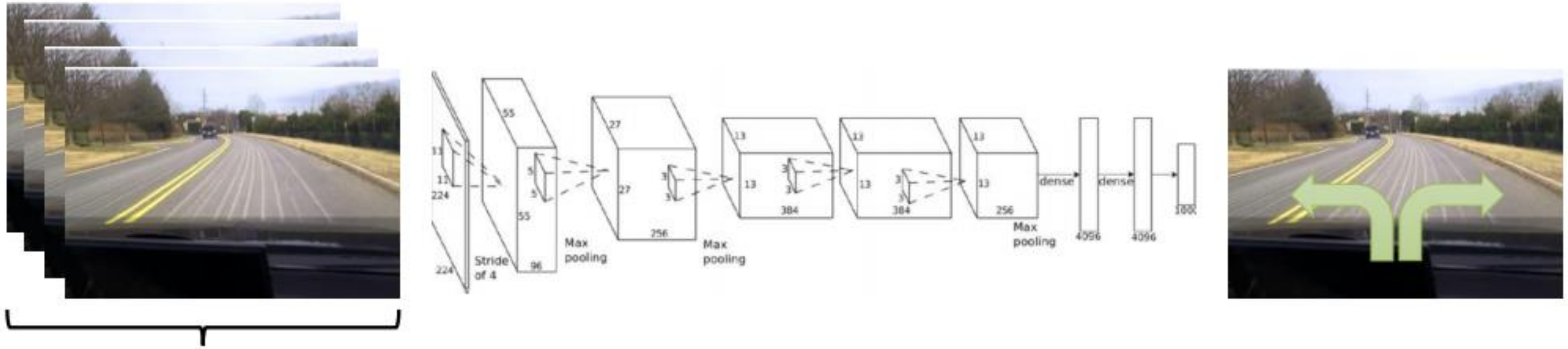


Can we make it work without more data?

- DAgger addresses the problem of distributional "drift"
- What if our model is so good that it doesn't drift?
  - Need to use the whole history of data (contextual action – deer encounters)
  - Need to mimic expert behavior very accurately but don't overfit



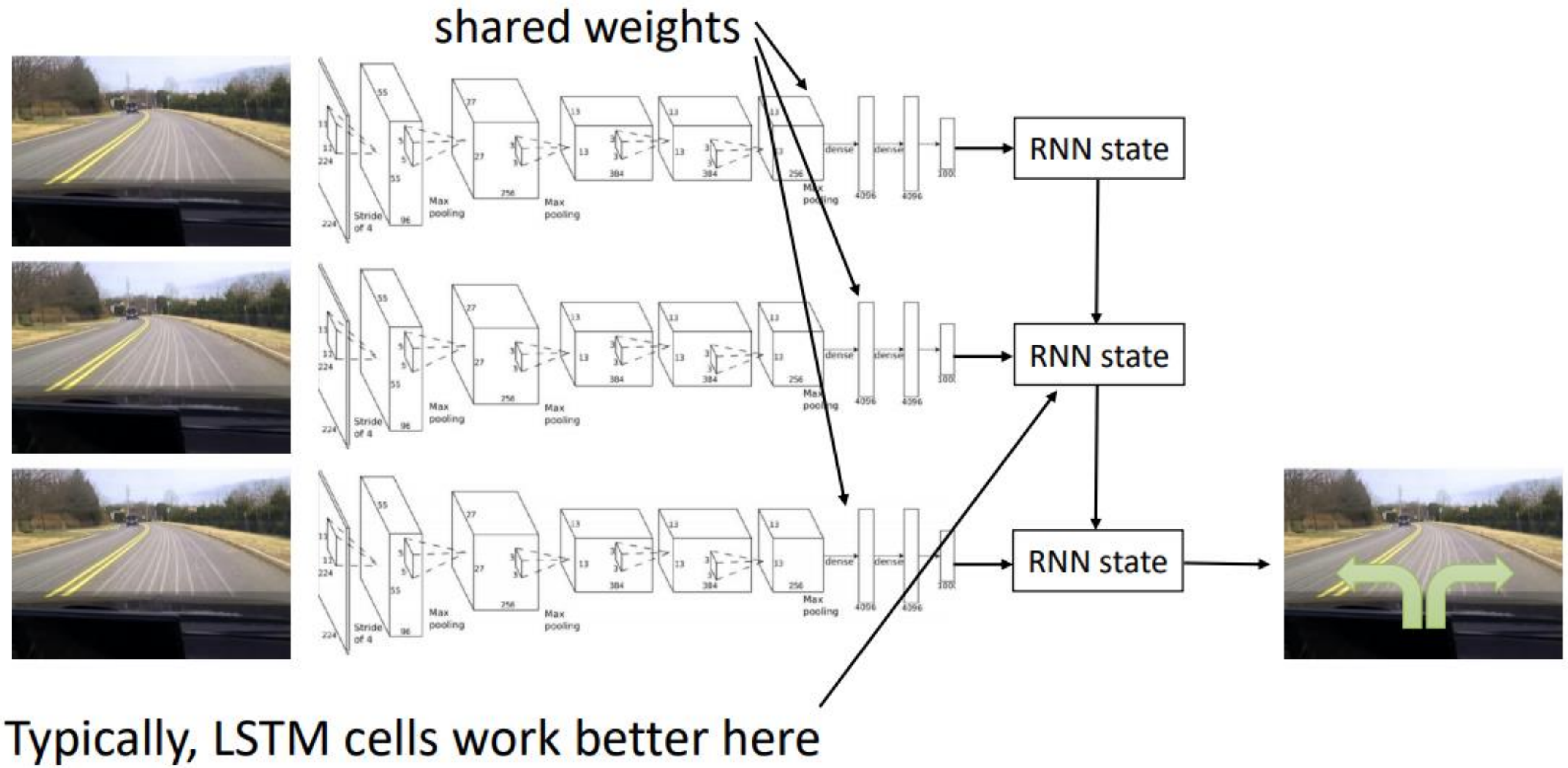
How can we use the whole history?



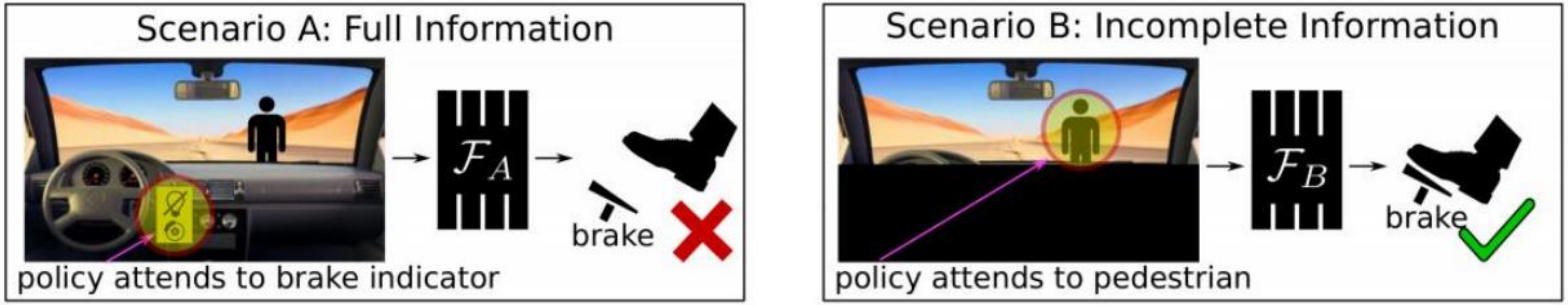
variable number of frames,
too many weights

Concatenating all observations +
weights into one observation to feed
into the model may cause overfitting

Can number of weights be reduced?



CONS: Incorporating History into the Model



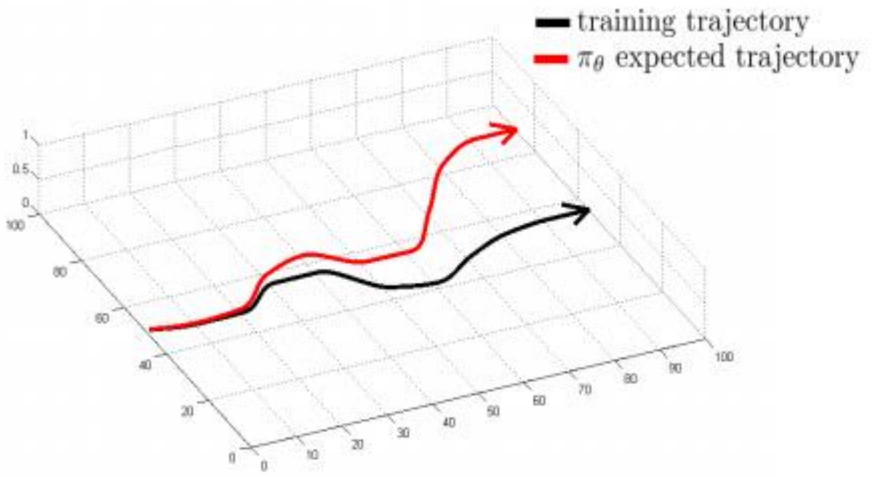
Haan et al. '19

<u>Causal Confusion</u>

- Consider a car that shows a light when the brake is stepped on
- Training data shows the brake is stepped on when an object is in front of the car (e.g., person, deer)
- Policy could learn: when there is a light, step on the brake
- Policy could learn: if the car behind you slows down, your car should slow down
- DAgger however mitigates causal confusion using human labeling (if model doesn't brake, human labeling in will result braking operation)

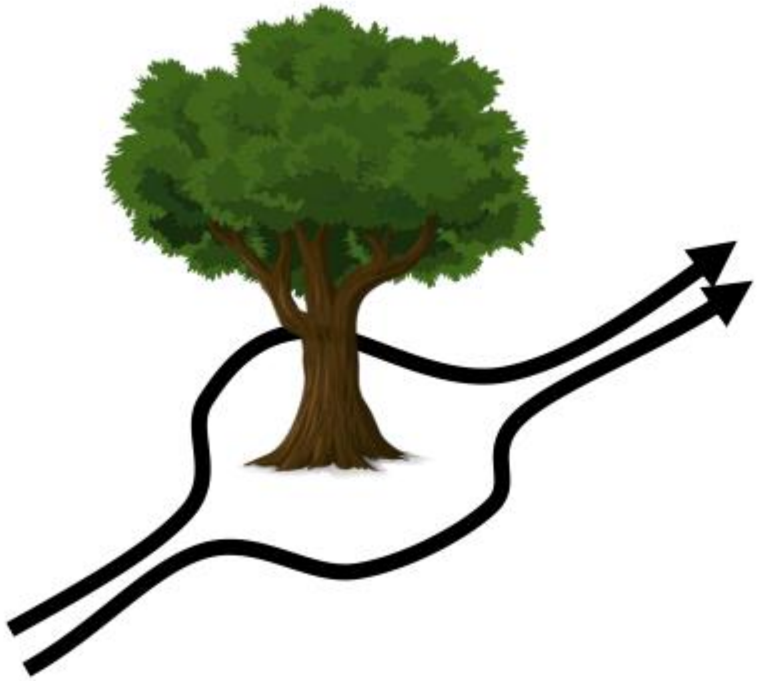
Can we make it work without more data?

- DAgger addresses the problem of distributional "drift"
- What if our model is so good that it doesn't drift?
  - Need to use the whole history of data (contextual action – deer encounters)
  - Need to mimic expert behavior very accurately but don't overfit



CONS:
Non-Markovian Behavior (of expert) &
Multi-modal Behavior

Multi-modal Behavior (given assumed unimodal behavior) Example



Goal: Traverse path without hitting tree (drone agent)

What if we averaged these two solutions?

Multi-modal Models

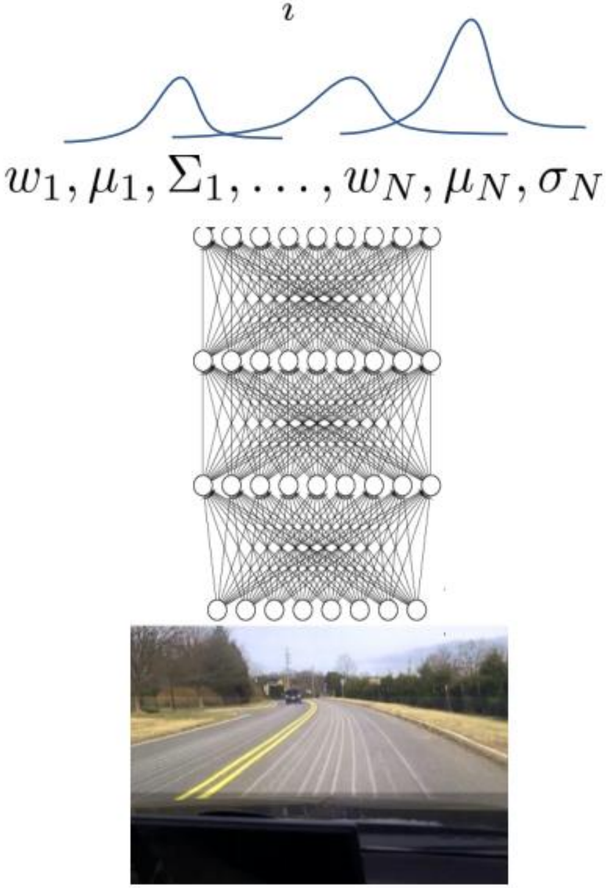
- Output mixture of Gaussians
- Latent variable models
- Autoregressive discretization

Multi-modal Models

- Output mixture of Gaussians
- Latent variable models
- Autoregressive discretization

approximate with i normal distributions (using i point clusters)

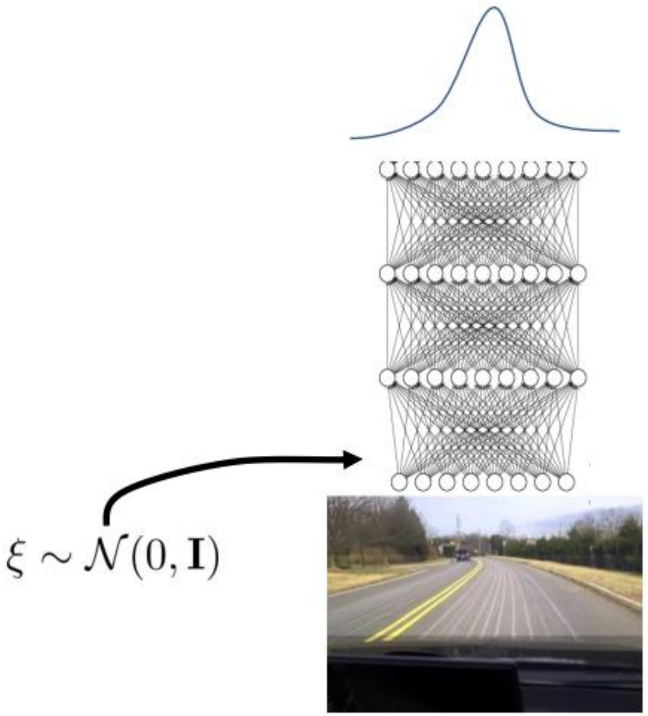
$$\pi(\mathbf{a}|\mathbf{o}) = \sum_i w_i \mathcal{N}(\mu_i, \Sigma_i)$$



Multi-modal Models

- Output mixture of Gaussians
- Latent variable models
- Autoregressive discretization

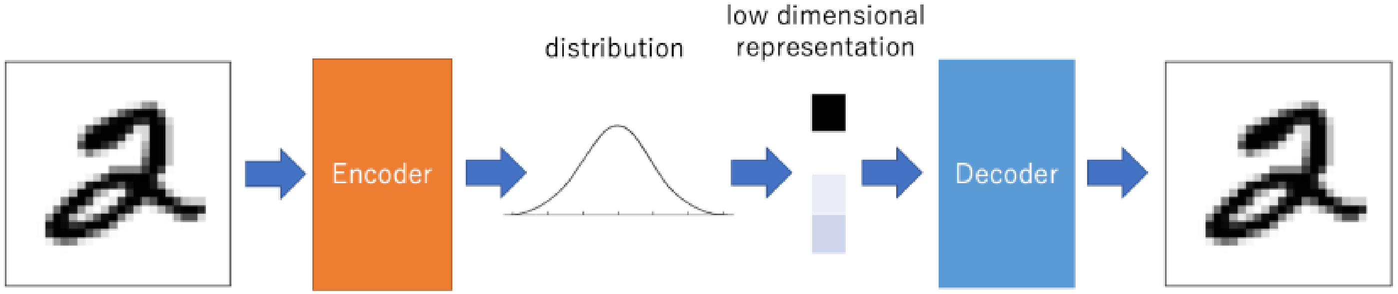
Relates observed variables to latent variables, such that each variable follows a univariate, normal distribution



Multi-modal Models

- Output mixture of Gaussians
- Latent variable models
  - Variational Autoencoder
- Autoregressive discretization

Variational Autoencoder



- encoder produces probability distribution (Gaussian in practice) over encodings
- to prevent encoder from learning a single value distribution, loss function: the KL divergence between the distribution produced by the encoder and a unit Gaussian distribution

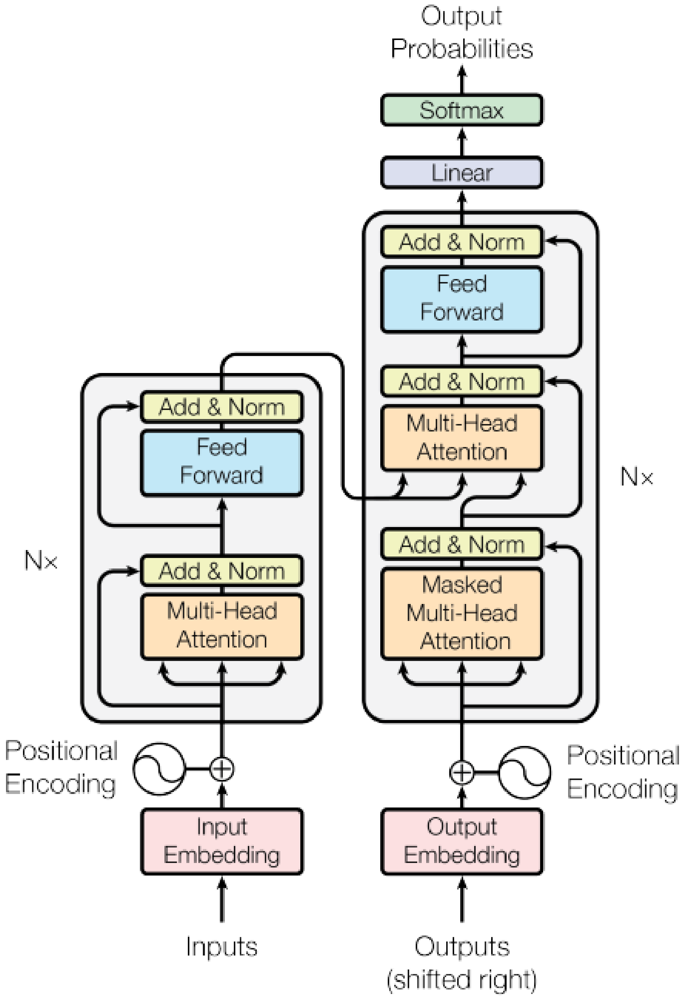


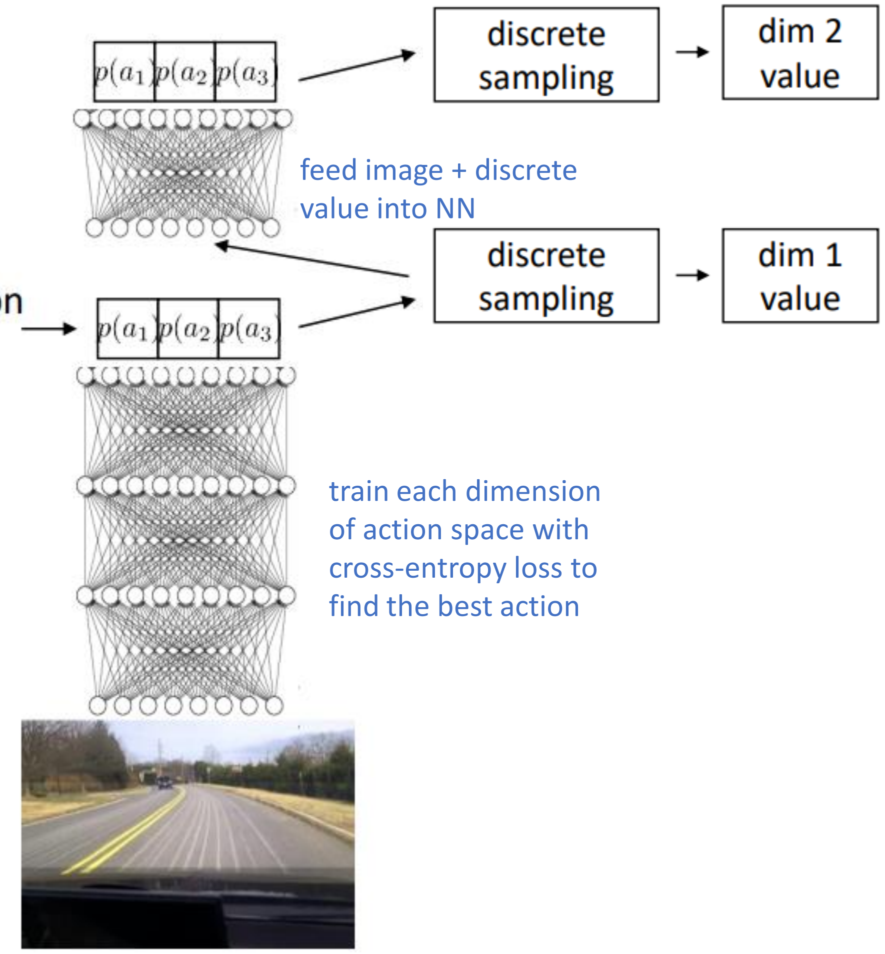
Figure 1: The Transformer - model architecture.

Multi-modal Models

- Output mixture of Gaussians
- Latent variable models
- Autoregressive discretization

(discretized) distribution over dimension 1 **only**

Possible actions for a choice of action (run: left, right, straight)



Low dimension action space (discrete or continuous)	→	discrete action space

in practice, increases exponentially as # action space increases

Imitation Learning Cost Function

$\mathbf{o}_t$
$\mathbf{a}_t$

training data

supervised learning

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

- states are distributed according to dynamics
- actions are distributed according to policy

function of number of mistakes made by learned policy

$$r(\mathbf{s},\mathbf{a}) = \log p(\mathbf{a} = \pi^\star(\mathbf{s})|\mathbf{s})$$

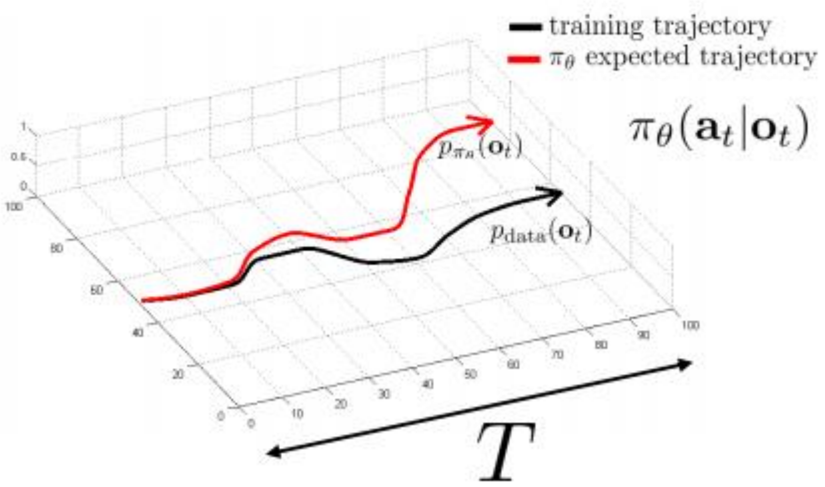
$$c(\mathbf{s},\mathbf{a}) = \begin{cases} 0 \text{ if } \mathbf{a} = \pi^\star(\mathbf{s}) \\ 1 \text{ otherwise} \end{cases}$$

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

Goal: $min_\theta E_{s_{1:T}, a_{1:T}}[\sum_t c(s_t, a_t)]$

Analysis on Mistakes Bound: Method 1

$$c(s,a) = \begin{cases} 0 \; if \; a = \pi^*(s) \\ 1 \; otherwise \end{cases}$$



$$assume\!: \; \pi_\theta(a \neq \pi^*(s)|s) \leq \epsilon \quad \forall s \in D_{train}$$

avoid mistakes on what you train on, bound ϵ for mistake

$$\rightarrow E[\pi_\theta(a \neq \pi^*(s)|s)] \leq \epsilon$$

states do not need to be fully observed, but ϵ error might be impossible if observations are too impoverished

first action is correct

$$E\left[\sum_t c(s_t, a_t)\right] \leq \underbrace{\boxed{\epsilon T} + \boxed{(1-\epsilon)}[\epsilon(T-1) + (1-\epsilon)[\ldots]]}_{T \text{ terms, each } O(\epsilon T)}$$

mistakes compound

$\boxed{O(\epsilon T^2)}$

Method 1 Shortcomings

- Somewhat unrealistic – if the policy makes a mistake once, it will keep making mistakes
  - The policy may still be able to generalize well for unseen states produced from a different underlying distribution than the training data: $p_{train}(s) \neq p_{\theta}(s)$

More General Analysis (Method 2)

$$c(s,a) = \begin{cases} 0 \; if \; a = \pi^*(s) \\ 1 \; otherwise \end{cases}$$

$assume$: $\pi_\theta(a \neq \pi^*(s)|s) \leq \epsilon$
$\forall s \in D_{train}, s \sim p_{train}(s)$

$\rightarrow E[\pi_\theta(a \neq \pi^*(s)|s)] \leq \epsilon$

Case 1

$with \; DAgger, p_{train}(s) = p_\theta(s)$:

$$E\left[\sum_t c(s_t, a_t)\right] \boxed{\leq \epsilon T}$$

Case 2

$p_{train}(s) \neq p_\theta(s)$:

$$p_\theta(s_t) = \underbrace{(1-\epsilon)^t}_{\text{all actions chosen correctly}} \underbrace{p_{train}(s_t)}_{\text{new state indistinguishable from expert's because policy has taken all correct actions}} + (1-(1-\epsilon)^t) \underbrace{p_{mistake}(s_t)}_{\text{policy finds itself in a state from some other distribution because it has made} \geq 1 \text{ mistake and veered off}}$$

(useful - next slide)

$$|p_\theta(s_t) - p_{train}(s_t)| = \sum_{s_t} |p_\theta(s_t) - p_{train}(s_t)|$$

$$p_\theta(s_t) = (1-\epsilon)^t p_{train}(s_t) + (1-\ (1-\epsilon)^t) p_{mistake}(s_t)$$

$$p_{train}(s_t) = (1-\epsilon)^t p_{train}(s_t) + (1-\ (1-\epsilon)^t) p_{train}(s_t)$$

$$= (1-\ (1-\epsilon)^t)|p_{mistake}(s_t) - p_{train}(s_t)|$$
$$\leq 2(1-\ (1-\epsilon)^t)$$
$$\boxed{\leq 2\epsilon t} \text{ (see identity below)}$$

useful identity: $(1-\epsilon)^t \geq 1 - \epsilon t$ for $\epsilon \in [0,1]$

More General Analysis (Method 2)

Want to find a bound for: $\sum_t E_{p_\theta(s_t)}[c_t] = \sum_t \sum_{s_t} p_\theta(s_t) c_t(s_t)$

$$\leq \sum_t \sum_{s_t} p_{train}(s_t) c_t(s_t) + |p_\theta(s_t) - p_{train}(s_t)| c_{max}(s_t) \quad [1]$$

$$c(s,a) = \begin{cases} 0 \; if \; a = \pi^*(s) \\ \underbrace{\boxed{1}}_{c_{max}(s_t)} \; otherwise \end{cases}$$

$$[1] \leq \sum_t (\epsilon + 2\epsilon T) \rightarrow \boxed{O(\epsilon T^2)}$$

Proof of [1]:

$$p_\theta(s_t) = p_{train}(s_t) + p_\theta(s_t) - p_{train}(s_t)$$
$$p_\theta(s_t) c_t(s_t) = p_{train}(s_t) c_t(s_t) + |p_\theta(s_t) - p_{train}(s_t)| c_t(s_t)$$
$$\leq p_{train}(s_t) c_t(s_t) + |p_\theta(s_t) - p_{train}(s_t)| c_{max}(s_t)$$

Behavioral cloning is not scalable
(quadratic increase in timesteps)

Content

- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

Definitions: Markov Chain

Andrey Markov

Markov chain

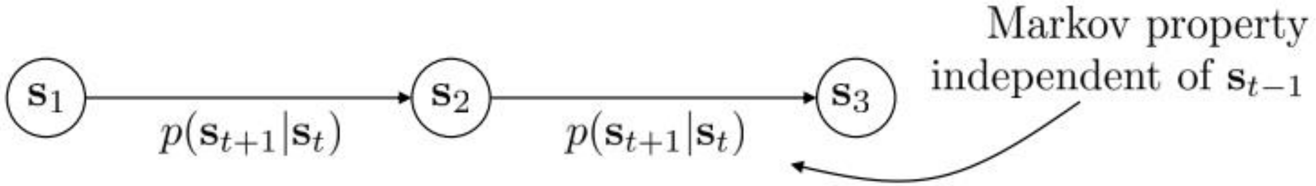
$\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$

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

$\mathcal{T}$ – transition operator — $p(s_{t+1}|s_t)$

why "operator"? — let $\mu_{t,i} = p(s_t = i)$ — $\vec{\mu}_t$ is a vector of probabilities

let $\mathcal{T}_{i,j} = p(s_{t+1} = i | s_t = j)$ — then $\vec{\mu}_{t+1} = \mathcal{T}\vec{\mu}_t$



Definitions: Markov Decision Process

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)

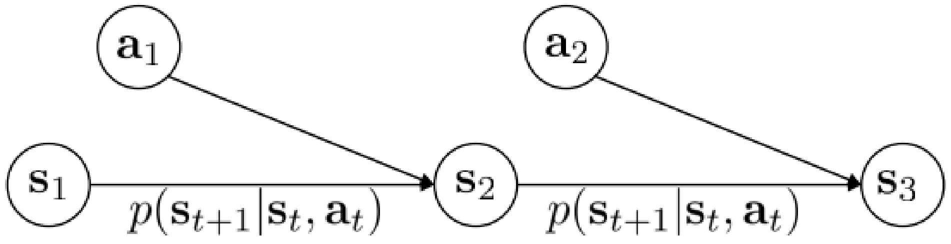
$\mathcal{T}$ – transition operator (now a tensor!)

let $\mu_{t,j} = p(s_t = j)$

let $\xi_{t,k} = p(a_t = k)$

let $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$



Definitions: Markov Decision Process

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!)

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

$r(s_t, a_t)$ – reward

Definitions: Partially Observed MDP

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)

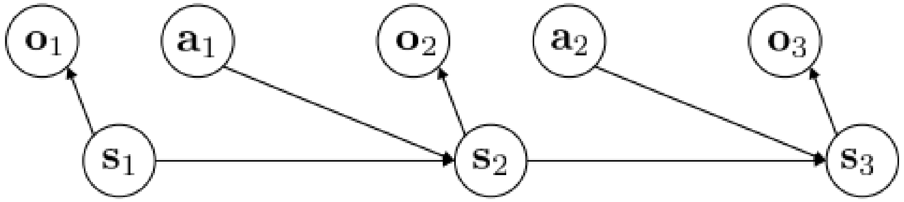
$\mathcal{A}$ – action space — actions $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{O}$ – 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 — $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$



Markov Chain vs. Markov Process

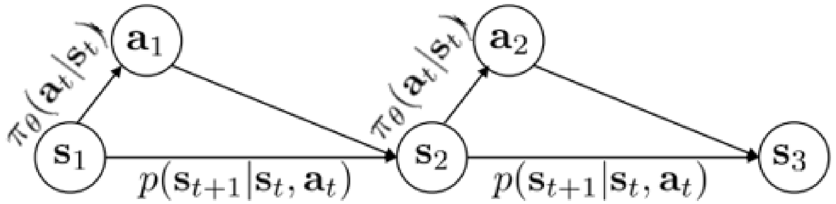
- Markov Chain:
  - discrete state space
  - discrete/continuous time parameter
  - stochastic process possesses Markov Property:

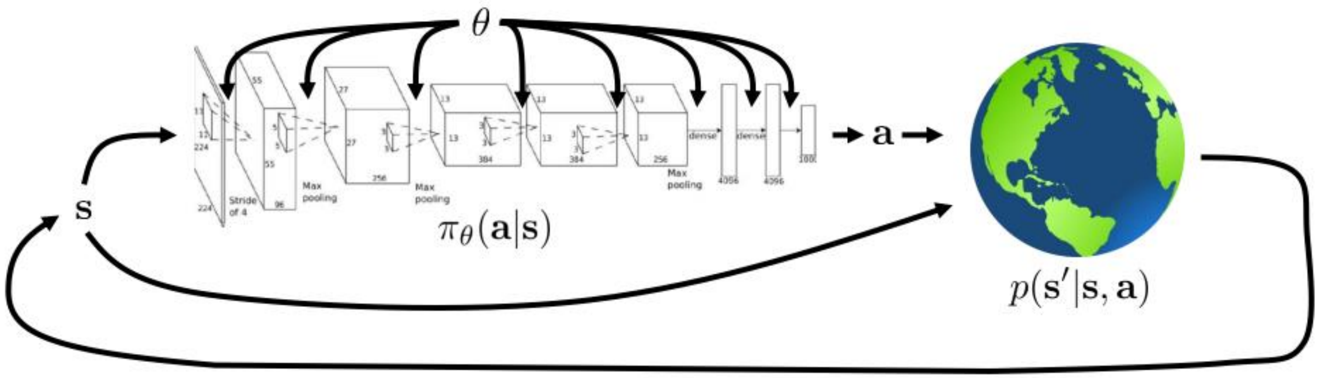
$$p(s_{t+1}|s_t, s_{t-1}, \dots, s_0) = p(s_{t+1}|s_t)$$

- Markov Process:
  - discrete/continuous state space
  - discrete/continuous time parameter
  - stochastic process possesses Markov Property:

$$p(s_{t+1}|s_t, s_{t-1}, \dots, s_0) = p(s_{t+1}|s_t)$$

RL in the context of MDPs

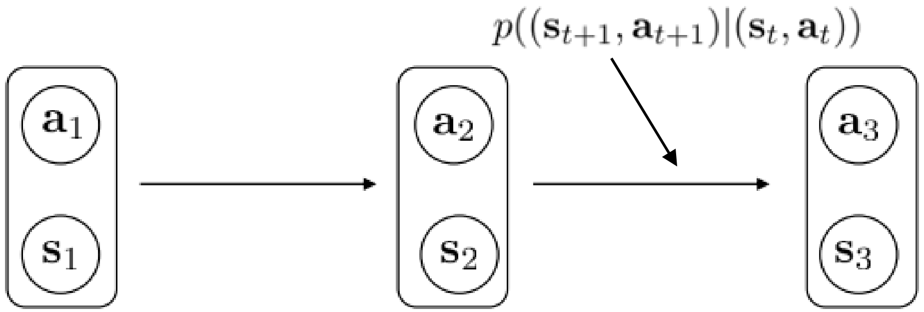




$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

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

$$p((\mathbf{s}_{t+1}, \mathbf{a}_{t+1})|(\mathbf{s}_t, \mathbf{a}_t)) =$$
$$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$$



Stationary Distribution

what if $T = \infty$? $\theta^\star = \arg\max_\theta \frac{1}{T} \sum_{t=1}^{T} E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_\theta(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)] \rightarrow E_{(\mathbf{s}, \mathbf{a}) \sim p_\theta(\mathbf{s}, \mathbf{a})}[r(\mathbf{s}, \mathbf{a})]$

does $p(\mathbf{s}_t, \mathbf{a}_t)$ converge to a *stationary* distribution?

$\mu = \mathcal{T}\mu$

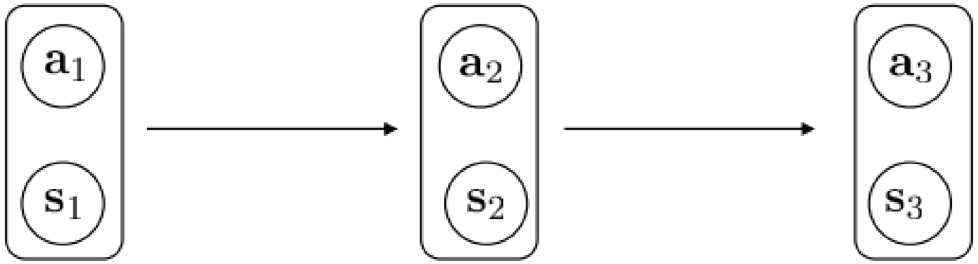
stationary = the same before and after transition

$(\mathcal{T} - \mathbf{I})\mu = 0$

μ is eigenvector of $\mathcal{T}$ with eigenvalue 1!

(always exists under some regularity conditions)

$\mu = p_\theta(\mathbf{s}, \mathbf{a})$ stationary distribution



state-action transition operator

$$\begin{pmatrix} \mathbf{s}_{t+1} \\ \mathbf{a}_{t+1} \end{pmatrix} = \mathcal{T} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} \qquad \begin{pmatrix} \mathbf{s}_{t+k} \\ \mathbf{a}_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

RL often uses expectations

- Smoothens the function

$r(\mathbf{x})$ – *not* smooth

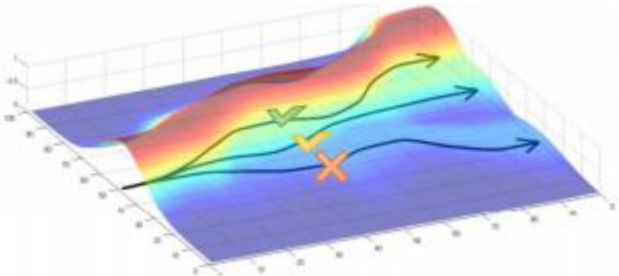
$\pi_\theta(\mathbf{a} = \text{fall}) = \theta$

$E_{\pi_\theta}[r(\mathbf{x})]$ – *smooth* in θ! $\quad (1-\theta) * (1) + \theta * (-1)$

Content

- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

RL Anatomy (w/ Ex. 1)



sum rewards at each timestep of trajectory

fit a model/ estimate the return

$$J(\theta) = E_\pi\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$$

as in DAgger Algorithm (trajectories in diagram)

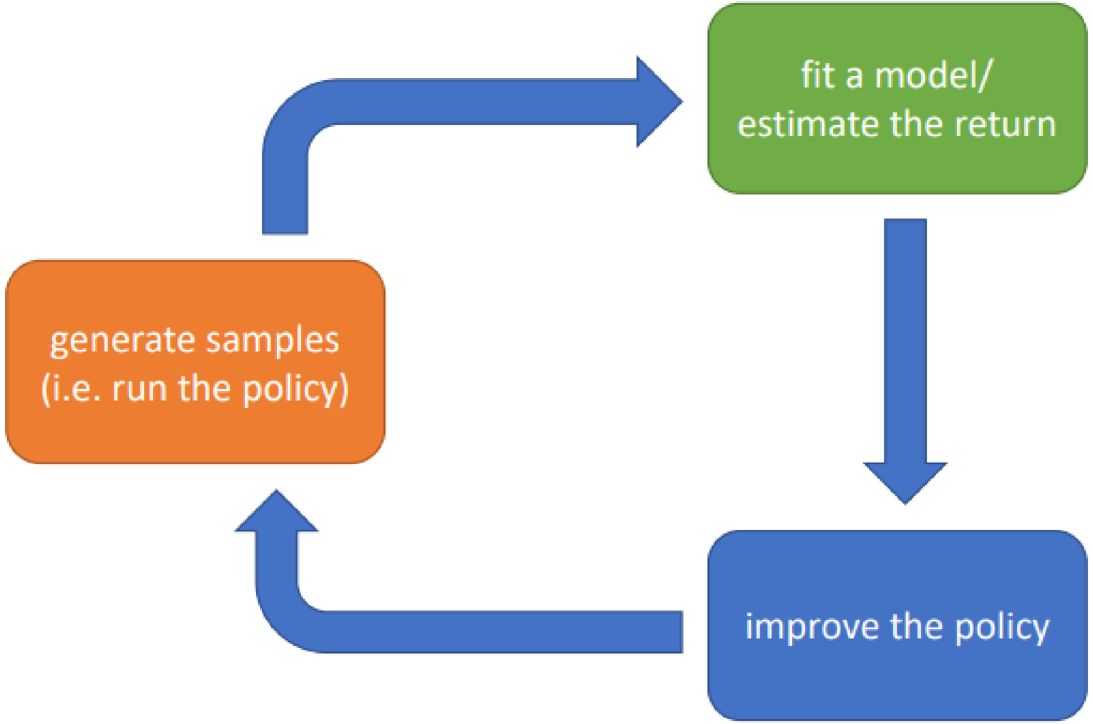
generate samples (i.e. run the policy)

improve the policy

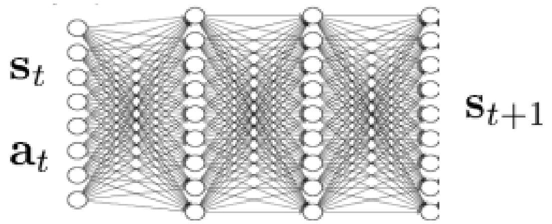
increase weight on trajectory with max rewards

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

RL Anatomy (w/ Ex. 2)

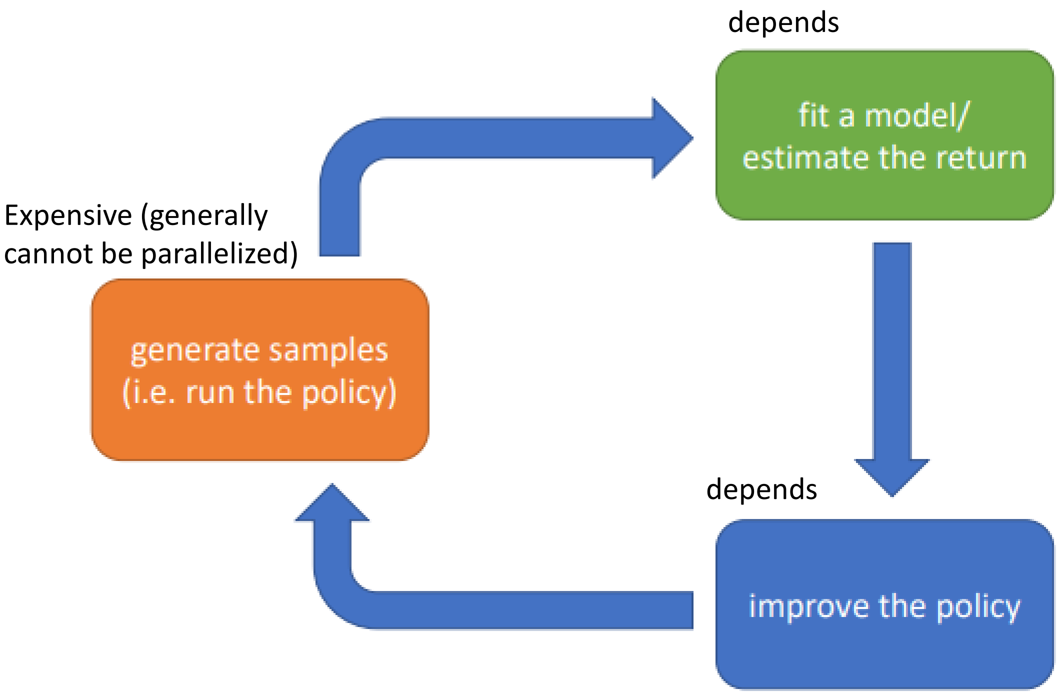


learn f_ϕ such that $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$



backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

The Bottleneck (Computational Efficiency)



$$J(\theta) = E_\pi\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$$

trivial, fast

learn $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$

expensive

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

Q-function and Value function

Q-function:

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right]$: Expected total reward from taking a_t in s_t

Value function:

$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t\right]$: Expected total reward from s_t

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

$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}[V^\pi(\mathbf{s}_1)]$ is the RL objective!

Using Q-functions and Value functions

Idea 1: if we have policy π, and we know $Q^\pi(\mathbf{s}, \mathbf{a})$, then we can *improve* π:

set $\pi'(\mathbf{a}|\mathbf{s}) = 1$ if $\mathbf{a} = \arg\max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$

this policy is at least as good as π (and probably better)!

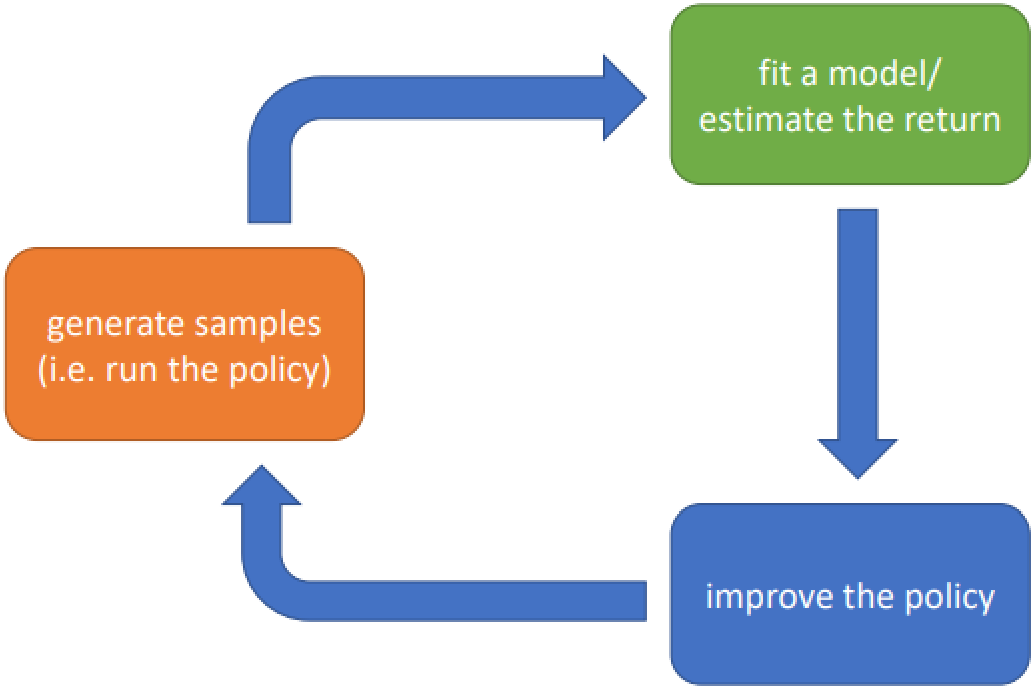
and it doesn't matter what π is

Idea 2: compute gradient to increase probability of good actions $\mathbf{a}$:

if $Q^\pi(\mathbf{s}, \mathbf{a}) > V^\pi(\mathbf{s})$, then $\mathbf{a}$ is *better than average* (recall that $V^\pi(\mathbf{s}) = E[Q^\pi(\mathbf{s}, \mathbf{a})]$ under $\pi(\mathbf{a}|\mathbf{s})$)

modify $\pi(\mathbf{a}|\mathbf{s})$ to increase probability of $\mathbf{a}$ if $Q^\pi(\mathbf{s}, \mathbf{a}) > V^\pi(\mathbf{s})$

Revisiting RL Anatomy



this often uses Q-functions or value functions

Content

- Introduction to Reinforcement Learning
  - Definitions & Motivation
    - Deep Learning (DL)
    - Reinforcement Learning (RL)
    - Inverse Reinforcement Learning (inverse RL)
    - Deep Reinforcement Learning (deep RL)
    - Transfer Learning & Meta-learning
    - Connection to the human brain
    - Forms of Supervision
- Imitation Learning/Behavioral Cloning
  - DAgger Algorithm (Algorithm + Analysis)
- Markov Decision Process (MDP)
- RL Algorithm Anatomy
- RL Algorithm Types + Comparison

Note:

- Following slides are for introduction/overview only
- List is not exhaustive
- Each algorithm type is dedicated to future individual lectures (future weeks)

Model-based and Model-free RL

- Model-based RL: uses the environment, action and reward to get the most reward from the action
- Model-free RL: does not use the environment, but only uses action and reward to learn the action that results in the best reward

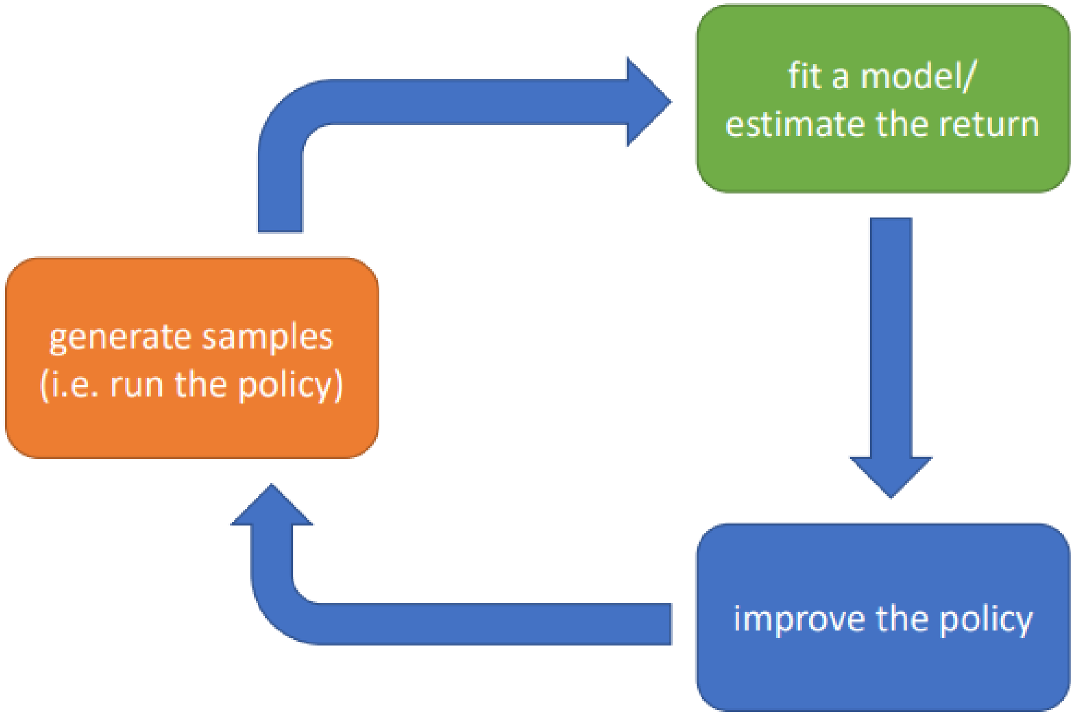
RL Algorithm Types (Model-free)

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

all use this objective function

- Policy gradients (doesn’t need full observability):
  - directly different objective and use it for gradient descent
- Value-based (needs full observability – Markovian state):
  - estimate value function or Q-function of the optimal policy (no explicit policy – only keep track of Q & V)
- Actor-critic:
  - calculate the gradient using Q or V to improve policy (Q or V + policy gradients)
  - can be off-policy or on-policy
    - off-policy learner learns the value of the optimal policy independently of the agent's actions (offline learning)
    - on-policy learner learns the value of the current policy being carried out by the agent including the exploration steps; need to generate new samples per policy change (online learning)

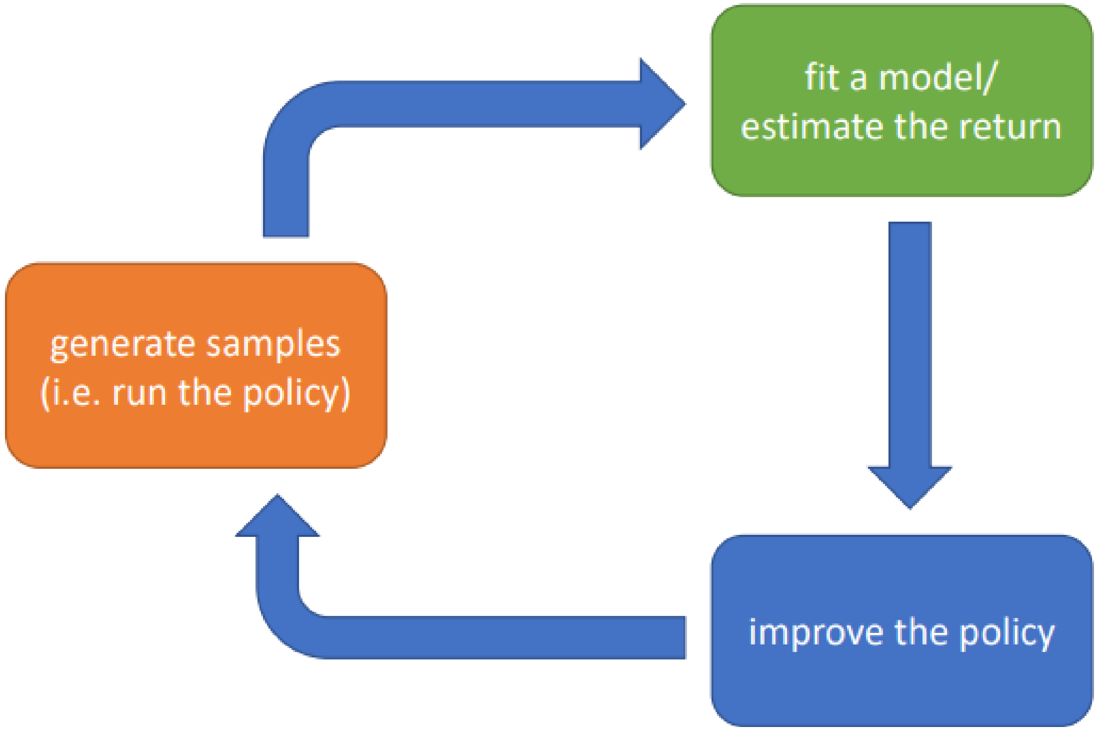
Direct policy gradients



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

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

Actor-critic: value functions + policy gradients



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

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

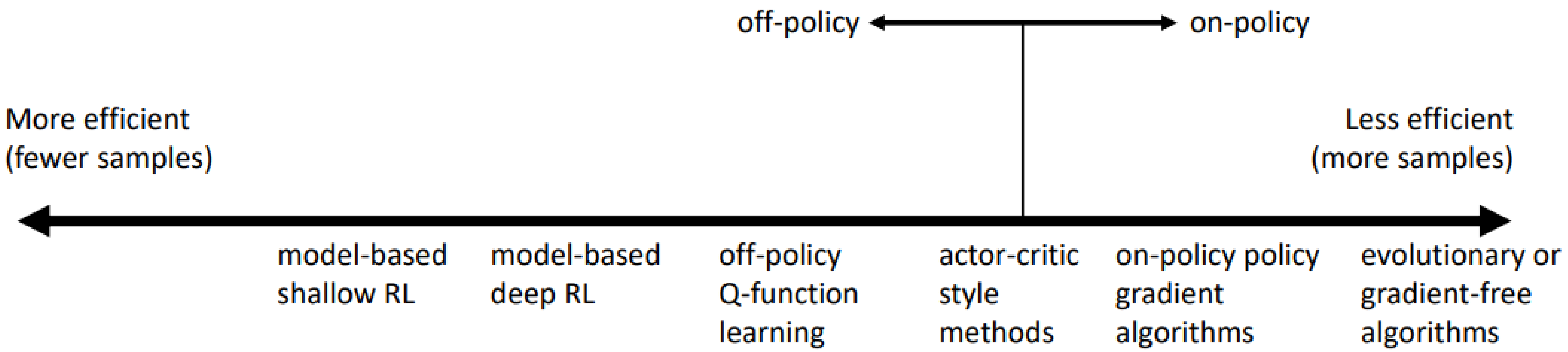
Model-based RL Algorithms

- Just use the model to plan (no policy)
  - Trajectory optimization/optimal control (primarily in continuous spaces) – essentially backpropagation to optimize over actions
- Backpropagate gradients into the policy
- Use the model to learn a value function
  - Dynamic programming
  - Generate simulated experience for model-free learner

RL Algorithm Design

- Different tradeoffs
  - Sample efficiency (# times to run policy for samples)
  - Stability & ease of use (frequency of unknown situation encounters)
- Different assumptions
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Converges to stationary distribution (RL doesn't require this –more later)?
- Different things are easy or hard in different settings
  - Easier to represent the policy?
    - physics of environment is complicated, but pattern for optimal behavior is simple
  - Easier to represent the model?
    - world is simple, policy is complicated (e.g. chess game)

Comparison: sample efficiency



Comparison: stability and ease of use

Does it converge?

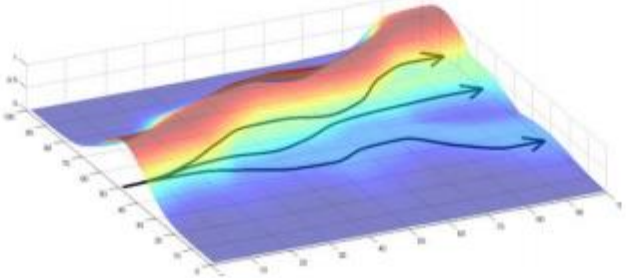
And if it converges, to what?

And does it converge every time?

- Supervised learning: almost always gradient descent
- Reinforcement learning: often not gradient descent
  - Q-learning: fixed point iteration
  - Model-based RL: model is not optimized for expected reward

Comparison: assumptions

- Common assumption #1: full observability
  - Generally assumed by Q or V function based methods (since model-free)
  - Can be mitigated by adding recurrence
- Common assumption #2: episodic learning (learning separated in different steps/components)
  - 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



References

- https://www.cs.cmu.edu/~sross1/publications/Ross-AIStats11-NoRegret.pdf (Ross et al., 2011)
- https://arxiv.org/abs/1604.07316 (Bojarski et al., 2016 Nvidia)
- https://arxiv.org/abs/1905.11979 (Haan et al., 2019)
- UC Berkeley CS 285 HW assignments: https://github.com/berkeleydeeprlcourse/

Extra

Implementing Dagger

DAgger: Dataset Aggregation

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

Ross et al. '11

rl_trainer.py

```
def collect_training_trajectories(self, itr, load_initial_expertdata, collect_policy, batch_size):
    """
    :param itr:
    :param load_initial_expertdata:  path to expert data pkl file
    :param collect_policy:  the current policy using which we collect data
    :param batch_size:  the number of transitions we collect
    :return:
        paths: a list trajectories
        envsteps_this_batch: the sum over the numbers of environment steps in paths
        train_video_paths: paths which also contain videos for visualization purposes
    """

    # TODO decide whether to load training data or use
    # HINT: depending on if it's the first iteration or not,
    # decide whether to either
    # load the data. In this case you can directly return as follows
    # ``` return loaded_paths, 0, None ```

    # collect data, batch_size is the number of transitions you want to collect.
    # use initial_expertdata for the first iteration
    if itr == 0:
        print(load_initial_expertdata)
        with open(load_initial_expertdata, "rb") as f:
            loaded_paths = pickle.load(f)
        return loaded_paths, 0, None
```

Implementing Dagger

DAgger: Dataset Aggregation

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

Ross et al. '11

utils.py

```python
def Path(obs, image_obs, acs, rewards, next_obs, terminals):
    """
        Take info (separate arrays) from a single rollout
        and return it in a single dictionary
    """
    if image_obs != []:
        image_obs = np.stack(image_obs, axis=0)
    return {"observation": np.array(obs, dtype=np.float32),
            "image_obs": np.array(image_obs, dtype=np.uint8),
            "reward": np.array(rewards, dtype=np.float32),
            "action": np.array(acs, dtype=np.float32),
            "next_observation": np.array(next_obs, dtype=np.float32),
            "terminal": np.array(terminals, dtype=np.float32)}
```

```python
def sample_trajectory(env, policy, max_path_length, render=False, render_mode=('rgb_array')):
    # initialize env for the beginning of a new rollout
    ob = env.reset()  # HINT: should be the output of resetting the env

    # init vars
    obs, acs, rewards, next_obs, terminals, image_obs = [], [], [], [], [], []
    steps = 0
    while True:

        # render image of the simulated env
        if render:
            if 'rgb_array' in render_mode:
                if hasattr(env, 'sim'):
                    image_obs.append(env.sim.render(camera_name='track', height=500, width=500)[::-1])
                else:
                    image_obs.append(env.render(mode=render_mode))
            if 'human' in render_mode:
                env.render(mode=render_mode)
                time.sleep(env.model.opt.timestep)

        # use the most recent ob to decide what to do
        obs.append(ob)
        ac = policy.get_action(ob)  # HINT: query the policy's get_action function
        ac = ac[0]
        acs.append(ac)

        # take that action and record results
        ob, rew, done, _ = env.step(ac)

        # record result of taking that action
        steps += 1
        next_obs.append(ob)
        rewards.append(rew)

        # TODO end the rollout if the rollout ended
        # HINT: rollout can end due to done, or due to max_path_length
        rollout_done = (steps == max_path_length) or done  # HINT: this is either 0 or 1
        terminals.append(rollout_done)

        if rollout_done:
            break

    return Path(obs, image_obs, acs, rewards, next_obs, terminals)
```

Implementing Dagger

DAgger: Dataset Aggregation

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. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Ross et al. '11

rl_trainer.py, MLP_policy.py

```python
def do_relabel_with_expert(self, expert_policy, paths):
    print("\nRelabelling collected observations with labels from an expert policy...")

    # TODO relabel collected obsevations (from our policy) with labels from an expert policy
    # HINT: query the policy (using the get_action function) with paths[i]["observation"]
    # and replace paths[i]["action"] with these expert labels
    for i in range(len(paths)):
        paths[i]["action"] = expert_policy.get_action(paths[i]["observation"])
    return paths
```

obtain from
expert dictionary

```python
# query this policy with observation(s) to get selected action(s)
def get_action(self, obs):

    if len(obs.shape)>1:
        observation = obs
    else:
        observation = obs[None]

    # TODO return the action that the policy prescribes
    # HINT1: you will need to call self.sess.run
    # HINT2: the tensor we're interested in evaluating is self.sample_ac
    # HINT3: in order to run self.sample_ac, it will need observation fed into the feed_dict
    return self.sess.run([self.sample_ac], feed_dict={self.observations_pl: observation})[0]
```

Implementing Dagger

DAgger: Dataset Aggregation

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

Ross et al. '11

rl_trainer.py, replay_buffer.py

```
def add_rollouts(self, paths, concat_rew=True):

    # add new rollouts into our list of rollouts
    for path in paths:
        self.paths.append(path)

    # convert new rollouts into their component arrays, and append them onto our arrays
    observations, actions, rewards, next_observations, terminals = convert_listofrollouts(paths, concat_rew)

    if self.obs is None:
        self.obs = observations[-self.max_size:]
        self.acs = actions[-self.max_size:]
        self.rews = rewards[-self.max_size:]
        self.next_obs = next_observations[-self.max_size:]
        self.terminals = terminals[-self.max_size:]
    else:
        self.obs = np.concatenate([self.obs, observations])[-self.max_size:]
        self.acs = np.concatenate([self.acs, actions])[-self.max_size:]
        if concat_rew:
            self.rews = np.concatenate([self.rews, rewards])[-self.max_size:]
        else:
            if isinstance(rewards, list):
                self.rews += rewards
            else:
                self.rews.append(rewards)
            self.rews = self.rews[-self.max_size:]
        self.next_obs = np.concatenate([self.next_obs, next_observations])[-self.max_size:]
        self.terminals = np.concatenate([self.terminals, terminals])[-self.max_size:]
```

```
# add collected data to replay buffer
self.agent.add_to_replay_buffer(paths)
```

repeat DAgger

```
# train agent (using sampled data from replay buffer)
self.train_agent()  ## TODO implement this function below
```