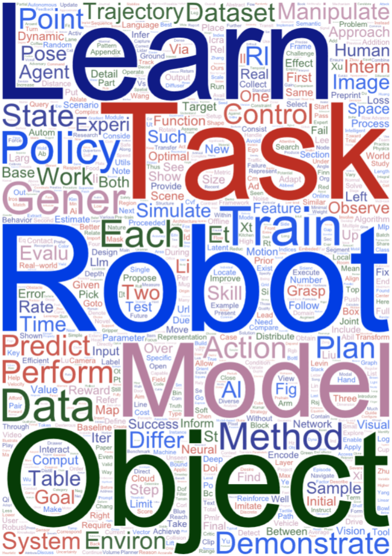


Robot Learning

Imitation Learning 2

Inspired by Guanya Shi's Lecture 6

Wolfgang Hönig
Technical University of Berlin
Summer 2024

Recap

- Imitation Learning
  - Given: expert demonstration data $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$
  - Goal: reproduce demonstrations
- Main Challenges:
  - Distributional Domain Shift   Solutions:
    - Behavior Cloning: add noise
    - DAgger: interactively add additional *expert* data
    - Trajectory Distribution Learning: rely on controller
  - Data Collection   Solutions:
    - Humans: teleoperation, kinesthetic teaching, motion capture, videos

Recap

- Imitation Learning
  - Given: expert demonstration data $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$
  - Goal: reproduce demonstrations
- Main Challenges:
  - Distributional Domain Shift   Solutions:
    - Behavior Cloning: add noise
    - DAgger: interactively add additional *expert* data
    - Trajectory Distribution Learning: rely on controller
  - Data Collection   Solutions:
    - Humans: teleoperation, kinesthetic teaching, motion capture, videos
    - **high-effort computations** (w.r.t. to computation or observation), e.g., *Privileged Teacher*

Outline Today

- Data Collection: Privileged Teacher
- Generative Models
- Case Studies
  - Quadrotor Acrobatics
  - Learning from ALOHA data
  - Transfer Learning

Privileged Teacher

- So far we considered to directly learn $\pi_\theta : x \mapsto u$ (or $\pi_\theta : y \mapsto u$)
- y might be high-dimensional or unstructured (e.g., RGBD sequences)
- Key insight: First learn *privileged* policy ("teacher"); use it to generate data for the "student"
    (i) Learn $\pi_{\theta_1} : z \mapsto u$ (where z contains some "ground truth" data, e.g., states, traffic lights, neighbor behavior)
    (ii) Use π_{θ_1} to generate data $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$
    (iii) Learn $\pi_{\theta_2} : x \mapsto u$

Privileged Teacher

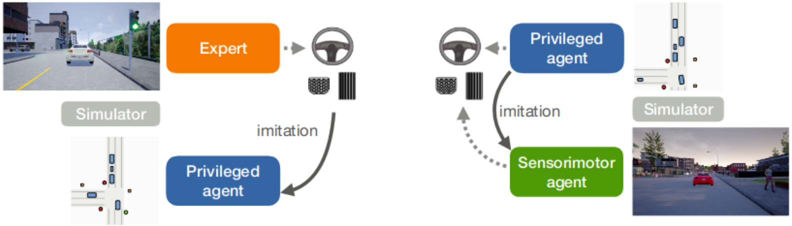
Learning by Cheating

Dian Chen UT Austin

Brady Zhou Intel Labs, UT Austin

Vladlen Koltun Intel Labs

Philipp Krähenbühl UT Austin



(a) Privileged agent imitates the expert (b) Sensorimotor agent imitates the privileged agent

https://youtu.be/u9ZCxxD-UUw

Privileged Teacher

- Pros and Cons compared to one-stage IL?

Privileged Teacher

- Pros and Cons compared to one-stage IL?

Pros:

- Second stage can be easily trained with DAgger
- Data augmentation simple

Cons

- Simulation-focused
- Hierarchical approach (requires domain knowledge)

Generative Models

- Generative Model:
  - Input: Data $D = \{d^i\}_{i=1}^n$
  - Learning: find distribution p_θ such that $d^i \sim p_\theta$
  - Inference: generate novel data $d^* \sim p_\theta$

Generative Models

- Generative Model:
  - Input: Data $D = \{d^i\}_{i=1}^n$
  - Learning: find distribution p_θ such that $d^i \sim p_\theta$
  - Inference: generate novel data $d^* \sim p_\theta$

- What generative models do you know?

Generative Models

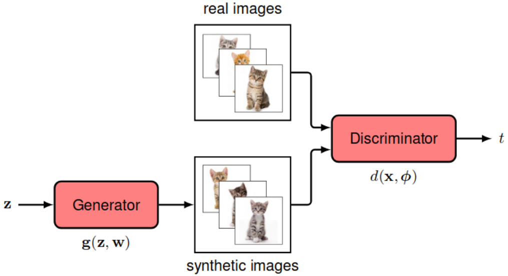
- Generative Model:
  - Input: Data $D = \{d^i\}_{i=1}^n$
  - Learning: find distribution p_θ such that $d^i \sim p_\theta$
  - Inference: generate novel data $d^* \sim p_\theta$

- What generative models do you know? [GAN, VAE, Diffusion, for details see:]

- Relationship to IL
  - If $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$, we can learn *conditional* distribution $p_\theta(u_t|x_t)$
  - Can also generate solution trajectories (esp. in combination with “classic” methods)

Generative Adverserial Network (GAN)

- Train two networks (generator and discriminator)



- Loss function (d_ϕ should be 1 for real data):

$$\max_\omega \min_\phi -\frac{1}{N_{data}} \sum_{n \in \text{data}} \ln d_\phi(x_n) - \frac{1}{N_{gen}} \sum_{n \in \text{gen}} \ln(1 - d_\phi(g_\omega(z_n)))$$

GAN + Imitation Learning = (GAIL)

Generative Adversarial Imitation Learning

Jonathan Ho
OpenAI
hoj@openai.com

Stefano Ermon
Stanford University
ermon@cs.stanford.edu

Algorithm 1 Generative adversarial imitation learning

Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
for $i = 0, 1, 2, \ldots$ **do**
    Sample trajectories $\tau_i \sim \pi_{\theta_i}$
    Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{17}$$

    Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

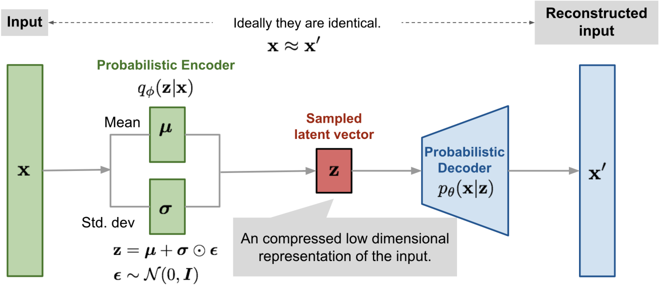
$$\begin{aligned} &\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta), \\ &\text{where } Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \end{aligned} \tag{18}$$

end for

- Generator is a policy $x \mapsto u$
- Discriminator has x, u as input
- Steps:
  - (i) **Rollout/Sample trajectories using generator (=policy)**
  - (ii) Update discriminator
  - (iii) Update policy

Variational Autoencoder (VAE)

- Train two networks (encoder and decoder)



ML Lecture, slides 8 and 9

- Loss function:

$$\min_{\theta,\phi} -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \mid p_\theta(\mathbf{z}))$$

Variational Autoencoder (VAE)

- Training: SGD Updates for both networks

repeat
  $\mathcal{L} \leftarrow 0$
  for $j \in \{1, \dots, M\}$ **do**
    $\epsilon_{nj} \sim \mathcal{N}(0, 1)$
    $z_{nj} \leftarrow \mu_j(\mathbf{x}_n, \phi)\epsilon_{nj} + \sigma_j^2(\mathbf{x}_n, \phi)$
    $\mathcal{L} \leftarrow \mathcal{L} + \frac{1}{2}\left\{1 + \ln \sigma_{nj}^2 - \mu_{nj}^2 - \sigma_{nj}^2\right\}$
  end for
  $\mathcal{L} \leftarrow \mathcal{L} + \ln p(\mathbf{x}_n | \mathbf{z}_n, \mathbf{w})$
  $\mathbf{w} \leftarrow \mathbf{w} + \eta \nabla_{\mathbf{w}} \mathcal{L}$ `// Update decoder weights`
  $\phi \leftarrow \phi + \eta \nabla_{\phi} \mathcal{L}$ `// Update encoder weights`
until converged
return $\mathbf{w}, \phi$

[There is an error in the Bishop book (Alg. 19.1): μ and σ are swapped at the highlighted line]

- Inference: Sample from Normal distribution and execute decoder

Variational Autoencoder (VAE) + Imitation Learning

2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Learning Sampling Distributions for Robot Motion Planning

Brian Ichter[*,1], James Harrison[*,2], Marco Pavone[1]

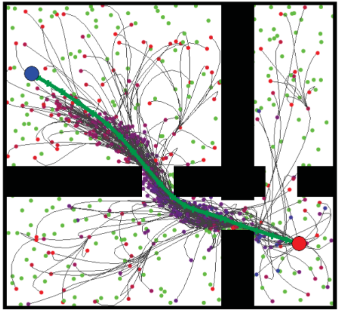
Learning Sample Distribution Methodology Outline

Offline:

1 **Input:** Data (successful motion plans, robot in action, human demonstration, etc.)
2 Construct conditioning variables y
3 Train CVAE, as in Fig. 2a

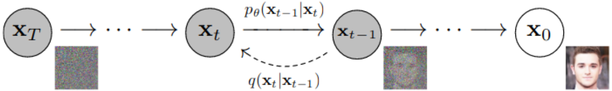
Online:

4 **Input:** Motion planning problem $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$, learned sample fraction λ
5 Construct conditioning variable y
6 Generate λN free samples from the CVAE latent space conditioned on y, as in Fig. 2b
7 Generate $(1-\lambda)N$ free samples from an auxiliary (uniform) sampler
8 Run sampling-based planner (e.g., PRM*, FMT*, RRT*)



Diffusion

- Train one network that "removes" noise



  Forward diffusion process: sample $\mathbf{x}_0$ and add iid Gaussian noise

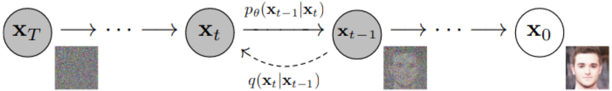
ML Lecture, slide 11

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Diffusion

- Train one network that “removes” noise



Reverse process: learn $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$

ML Lecture, slide 11

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Diffusion: Training

Algorithm 20.1: Training a denoising diffusion probabilistic model

Input: Training data $\mathcal{D} = \{\mathbf{x}_n\}$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Network parameters $\mathbf{w}$

---

for $t \in \{1, \ldots, T\}$ **do**
    $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$ `// Calculate alphas from betas`
end for
repeat
    $\mathbf{x} \sim \mathcal{D}$ `// Sample a data point`
    $t \sim \{1, \ldots, T\}$ `// Sample a point along the Markov chain`
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`
    $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1 - \alpha_t}\boldsymbol{\epsilon}$ `// Evaluate noisy latent variable`
    $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}\|^2$ `// Compute loss term`
    Take optimizer step
until converged
return $\mathbf{w}$

Diffusion: Sampling

Algorithm 20.2: Sampling from a denoising diffusion probabilistic model

Input: Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Sample vector $\mathbf{x}$ in data space

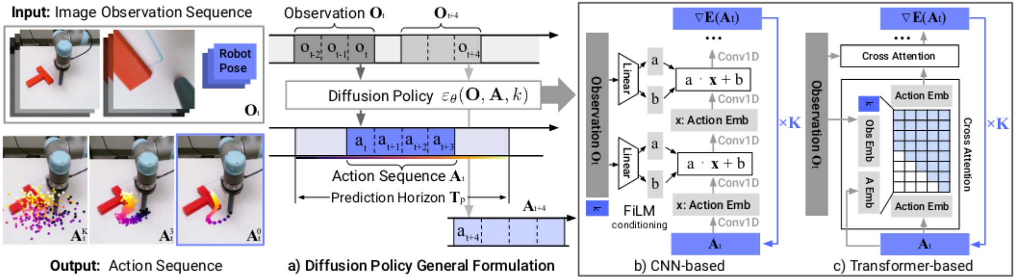
$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ // Sample from final latent space
for $t \in T, \ldots, 2$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$ // Calculate alpha
  // Evaluate network output
  $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}} \left\{ \mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} \mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) \right\}$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ // Sample a noise vector
  $\mathbf{z}_{t-1} \leftarrow \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$ // Add scaled noise
end for
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}} \left\{ \mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}} \mathbf{g}(\mathbf{z}_1, \mathbf{w}, t) \right\}$ // Final denoising step
return $\mathbf{x}$

Diffusion + Imitation Learning

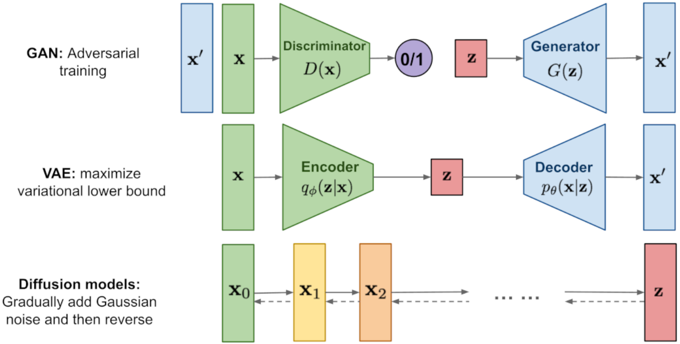
Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

Diffusion Policy:
Visuomotor Policy Learning via Action Diffusion

Cheng Chi[1], Siyuan Feng[2], Yilun Du[3], Zhenjia Xu[1], Eric Cousineau[2], Benjamin Burchfiel[2], Shuran Song[1]
[1] Columbia University [2] Toyota Research Institute [3] MIT
https://diffusion-policy.cs.columbia.edu



Comparison of Generative Models



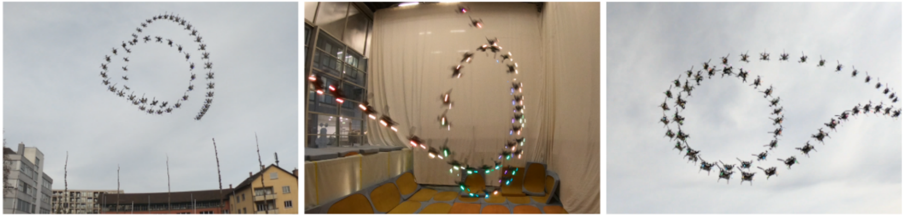
- What are advantages / disadvantages? (e.g., sample quality, sample efficiency, distribution “coverage”, ease of training)

Case Study: Deep Drone Acrobatics

Robotics: Science and Systems 2020
Corvalis, Oregon, USA, July 12-16, 2020

Deep Drone Acrobatics

Elia Kaufmann[*‡], Antonio Loquercio[*‡], René Ranftl[†], Matthias Müller[†], Vladlen Koltun[†], Davide Scaramuzza[‡]

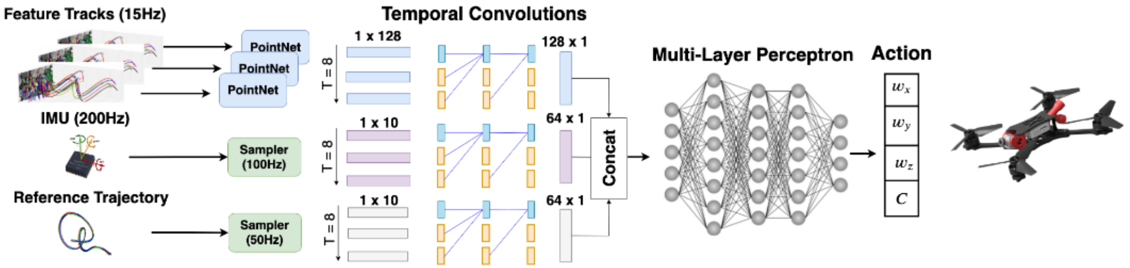


https://youtu.be/2N_wKXQ6MXA

Case Study: Deep Drone Acrobatics

- Input
  (i) **Abstraction** of sequence of last camera images (feature tracks)
  (ii) **Preprocessed** sequence of IMU data
  (iii) Reference trajectory
- Output
  - Desired body rates and thrust (to be tracked by attitude controller)
- Data
  - Purely from simulation (privileged expert = optimization-based MPC controller)
- Learning
  - Privileged Teacher (here: given, not learned from human demonstrations)
  - DAgger

Case Study: Deep Drone Acrobatics



Case Study: Deep Drone Acrobatics

Unique design choices:

- Pre-processing of input for **sim-to-real transfer**


- Asynchronous network branch inference
- Custom DAgger rollout for **sim-to-real transfer**: only use policy if similar to expert; also include random actions

Case Study: Using ALOHA Data

Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware

Tony Z. Zhao[1] Vikash Kumar[3] Sergey Levine[2] Chelsea Finn[1]
[1] Stanford University [2] UC Berkeley [3] Meta

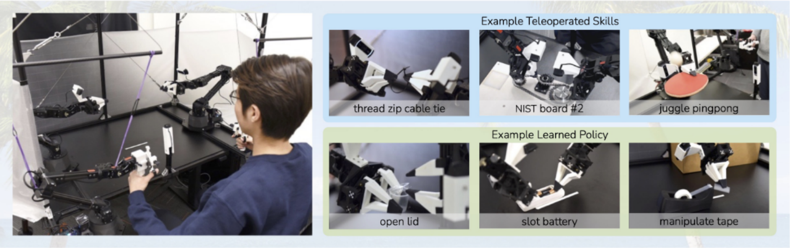
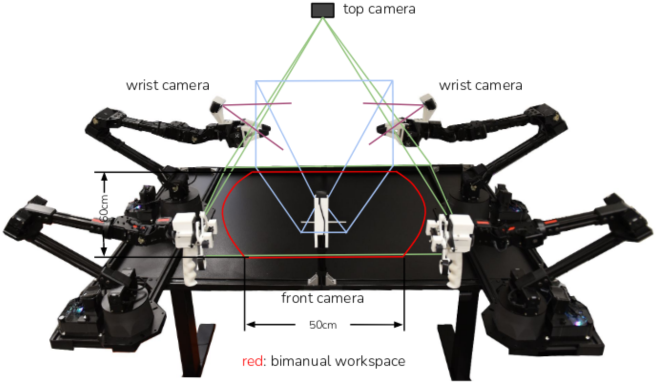
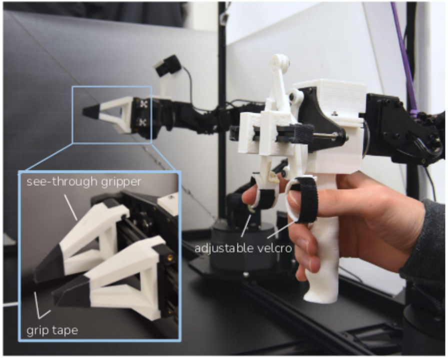


Fig. 1: *ALOHA : A Low-cost Open-source Hardware System for Bimanual Teleoperation*. The whole system costs <$20k with off-the-shelf robots and 3D printed components. *Left:* The user teleoperates by backdriving the leader robots, with the follower robots mirroring the motion. *Right: ALOHA* is capable of precise, contact-rich, and dynamic tasks. We show examples of both teleoperated and learned skills.

https://tonyzhaozh.github.io/aloha/

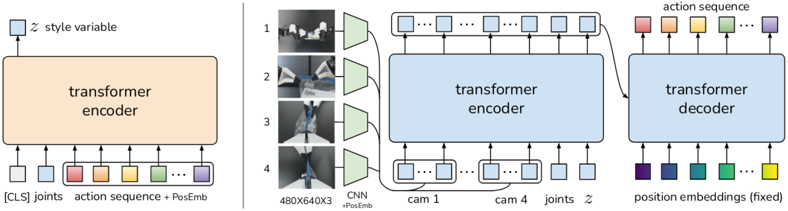
Case Study: Using ALOHA Data





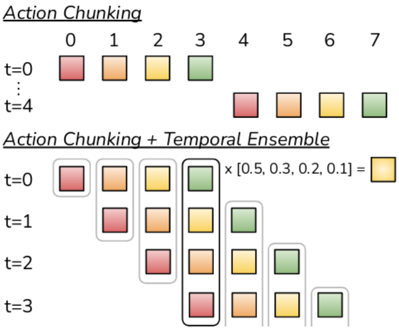
Case Study: Using ALOHA Data

- Conditional Variational Autoencoder (CVAE)
  - Encoder: joint positions, expert action sequence ($k >> 1$)
  - Latent space: z “style” (dim=32)
  - Decoder: observations (4 RGB images), joint positions, “style” z; output: planned action sequence



Case Study: Using ALOHA Data

- Inference: z is always set to 0 (deterministic generator)
- Key insights: transformer architectures for encoder and decoder; MPC-style encoding (action chunks + temporal ensemble)
- Fun statistics:
  - 80 M parameters; 5h training (RTX 2080 Ti); 10ms inference
  - 50 demonstrations per task (about 20min of data)

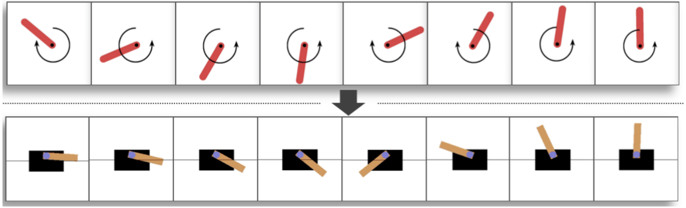


Case Study: Domain Adaptive Imitation Learning (DAIL)

Domain Adaptive Imitation Learning

Kuno Kim [1] Yihong Gu [2] Jiaming Song [1] Shengjia Zhao [1] Stefano Ermon [1]

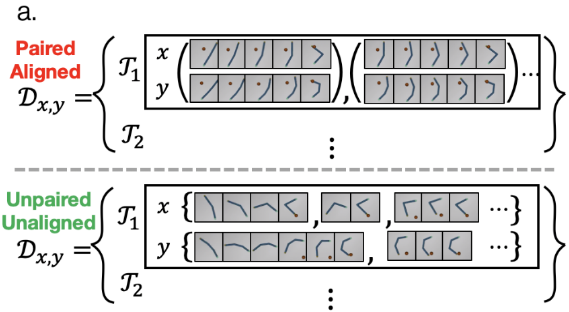
- How to perform a task, given demonstrations from a different domain (viewpoint, embodiment, and/or dynamics mismatch)?



https://youtu.be/l0tc1JCN_1M

Case Study: Domain Adaptive Imitation Learning (DAIL)

- Given: **unprocessed** examples for the same tasks for robots x and y
  - $D_{x,y} = \{(D_{M_x,T_i}, D_{M_y,T_i})\}_{i=1}^N$ for N tasks $\{T_i\}_{i=1}^N$
  - Data is not paired/aligned, i.e., $s_x^{(t)}$ does not "match" $s_y^{(t)}$



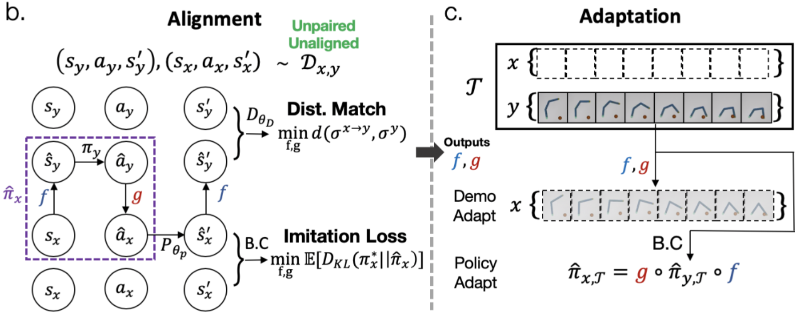
- Goal: Given a new demonstration of unseen task T_j for y, transfer/execute directly ("zero-shot") on robot x

Case Study: Domain Adaptive Imitation Learning (DAIL)

- Learning Alignment from $D_{x,y} = \{(D_{M_x,T_i}, D_{M_y,T_i})\}_{i=1}^N$:
  (i) Learn π^*_{y,T_i} for all T_i (Behavior Cloning)
  (ii) Learn mapping of states from x to y: $f_{\theta_f} : x_x \mapsto x_y$
  (iii) Learn mapping of actions from y to x: $g_{\theta_g} u_y \mapsto u_x$
  (iv) Learn dynamics/step function of x: $P^x_{\theta_P} : x_x, u_x \mapsto x_x$

Case Study: Domain Adaptive Imitation Learning (DAIL)

- Adaption
  (i) Learn π^*_{y,T_j} for new task T_j (Behavior Cloning)
  (ii) $\pi^*_{y,T_i}(x_x) = g_{\theta_g}(\pi^*_{y,T_j}(f_{\theta_f}(x_x)))$



Case Study: Domain Adaptive Imitation Learning (DAIL)

- Alignment Approach: Generative Adversarial MDP Alignment (GAMA)
  - Discriminator tries to separate real transitions $((x, u) \to x')$ from aligned transitions
  - "Generator" are f and g (deterministic)

Algorithm 1 Generative Adversarial MDP Alignment (GAMA)

input: Alignment task set $\mathcal{D}_{x,y} = \{(\mathcal{D}_{\mathcal{M}_{x,\mathcal{T}_i}}, \mathcal{D}_{\mathcal{M}_{y,\mathcal{T}_i}})\}_{i=1}^{N}$ of unpaired trajectories, fitted $\pi^*_{y,\mathcal{T}_i}$
while not done **do**:
for $i = 1, ..., N$ **do**:
Sample $(s_x, a_x, s'_x) \sim \mathcal{D}_{\mathcal{M}_{x,\mathcal{T}_i}}, (s_y, a_y, s'_y) \sim \mathcal{D}_{\mathcal{M}_{y,\mathcal{T}_i}}$ and store in buffer $\mathcal{B}^i_x, \mathcal{B}^i_y$
for $j = 1, ..., M$ **do**:
Sample mini-batch j from $\mathcal{B}^i_x, \mathcal{B}^i_y$
Update dynamics model with: $-\hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_P}(P^x_{\theta_P}(s_x, a_x) - s'_x)^2]$
Update discriminator: $\hat{\mathbb{E}}_{\pi^*_{y,\mathcal{T}_i}}[\nabla_{\theta^i_D} \log D_{\theta^i_D}(s_y, a_y, s'_y)] + \hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta^i_D} \log(1 - D_{\theta^i_D}(\hat{s}_y, \hat{a}_y, \hat{s}'_y))]$
Update alignments $(f_{\theta_f}, g_{\theta_g})$ with gradients:

$$-\hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_f} \log D_{\theta_D}(\hat{s}_y, \hat{a}_y, \hat{s}'_y)] + \hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_f}(\hat{\pi}_{x,\mathcal{T}_i}(s_x) - a_x)^2]$$

$$-\hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_g} \log D_{\theta_D}(\hat{s}_y, \hat{a}_y, \hat{s}'_y)] + \hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_g}(\hat{\pi}_{x,\mathcal{T}_i}(s_x) - a_x)^2]$$

Conclusion

- Imitation Learning works well for robotics
  - Efficient, effective, stable training
  - Fast inference
  - State-of-the-art real-robot results (mobile robots, manipulation, planning)
- Main challenge: acquire labeled data
  - Simulation possible (e.g., make slow algorithms fast) $\Rightarrow$ Use **DAgger** and/or **privileged teacher** paradigm
  - Only real data $\Rightarrow$ intuitive data collection interfaces, powerful generative and sequence models, transfer learning
- Details can be tricky (what to learn [policy, trajectory, value function], how to represent inputs, network architectures)
- Not discussed (yet): How to become better than the “expert” (notion of reward)