



MBRL

# BASICS

- Intuition: Habitual vs Goal-directed
- Learns an environment/dynamics model/network
- Function Approximation:  $P_\varphi: S \times A \rightarrow R \times S'$
- Model rollout:  $S_t \rightarrow A_t | \pi_\theta(S_t) \rightarrow R_t | P_\varphi(S_t, A_t) \rightarrow S_{t+1} | P_\varphi(S_t, A_t) \rightarrow \dots$

# MODEL-BASED CONTROL

- Model rollout from  $s_0$
- Objective:  $\min_{a_0 \dots a_{T-1}} \|s_{T-1} - s^*\|$  or  $\max_{a_0 \dots a_{T-1}} \sum_{t=0}^{T-1} r_t$

## MPC

- $s_0 \rightarrow a_0 | \pi_\theta(s_0) \rightarrow s_1 | P_\phi(s_0, a_0)$
- Backpropagate using objective  $\max_{a_1 \dots a_{T-1}} \sum_{t=1}^{T-1} r_t$  from model unrolling
- $s_0 \leftarrow s_1$
- Repeat

## ALEATORIC VS EPISTEMIC UNCERTAINTY

- Aleatoric def: Uncertainty due to system/environment(Latin: game of chance)
- Epistemic def: Uncertainty due to insufficient/biased data
- Compounding error(Long term)
- Probabilistic ensemble networks(Gaussian)
- E.g. PETs & MBPO

---

**Algorithm 2** Model-Based Policy Optimization with Deep Reinforcement Learning

---

- 1: Initialize policy  $\pi_\phi$ , predictive model  $p_\theta$ , environment dataset  $\mathcal{D}_{\text{env}}$ , model dataset  $\mathcal{D}_{\text{model}}$
  - 2: **for**  $N$  epochs **do**
  - 3:   Train model  $p_\theta$  on  $\mathcal{D}_{\text{env}}$  via maximum likelihood
  - 4:   **for**  $E$  steps **do**
  - 5:     Take action in environment according to  $\pi_\phi$ ; add to  $\mathcal{D}_{\text{env}}$
  - 6:     **for**  $M$  model rollouts **do**
  - 7:       Sample  $s_t$  uniformly from  $\mathcal{D}_{\text{env}}$
  - 8:       Perform  $k$ -step model rollout starting from  $s_t$  using policy  $\pi_\phi$ ; add to  $\mathcal{D}_{\text{model}}$
  - 9:     **for**  $G$  gradient updates **do**
  - 10:      Update policy parameters on model data:  $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$
-

# OFF-POLICY RL

## DEFINITION

- Learns value of the optimal policy disregarding agent's actions
- Behavior policy(Correlated with current)
- On policy algos: SARSA, policy gradient methods
- Off policy algos: Q-learning, soft a2c, DDPG

## DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

- **Objective:**  $\max_{\theta} \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [\sum_{t=0}^{T-1} Q(s_t, a_t)]$
- $\nabla_{\theta} \mathbb{E}[\sum_{t=0}^{T-1} Q(s_t, a_t)] = \mathbb{E}[\sum_{t=0}^{T-1} \nabla_{a_t} Q(s_t, a_t) \nabla_{\theta} a_t]$
- Learns from experience tuples in a buffer(Not fixed)
- Extrapolation error: Distinguish good (s,a)s from the bad ones(states visited by policy).
- Not truly off-policy

---

**Algorithm 1** Deep Deterministic Policy Gradient

---

```

1: Input: initial policy parameters  $\theta$ , Q-function parameters  $\phi$ , empty replay buffer  $\mathcal{D}$ 
2: Set target parameters equal to main parameters  $\theta_{\text{targ}} \leftarrow \theta$ ,  $\phi_{\text{targ}} \leftarrow \phi$ 
3: repeat
4:   Observe state  $s$  and select action  $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$ , where  $\epsilon \sim \mathcal{N}$ 
5:   Execute  $a$  in the environment
6:   Observe next state  $s'$ , reward  $r$ , and done signal  $d$  to indicate whether  $s'$  is terminal
7:   Store  $(s, a, r, s', d)$  in replay buffer  $\mathcal{D}$ 
8:   If  $s'$  is terminal, reset environment state.
9:   if it's time to update then
10:    for however many updates do
11:      Randomly sample a batch of transitions,  $B = \{(s, a, r, s', d)\}$  from  $\mathcal{D}$ 
12:      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

13:      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s, a, r, s', d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$$

14:      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

15:      Update target networks with

$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho \phi_{\text{targ}} + (1 - \rho) \phi \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho) \theta \end{aligned}$$

16:    end for
17:  end if
18: until convergence

```

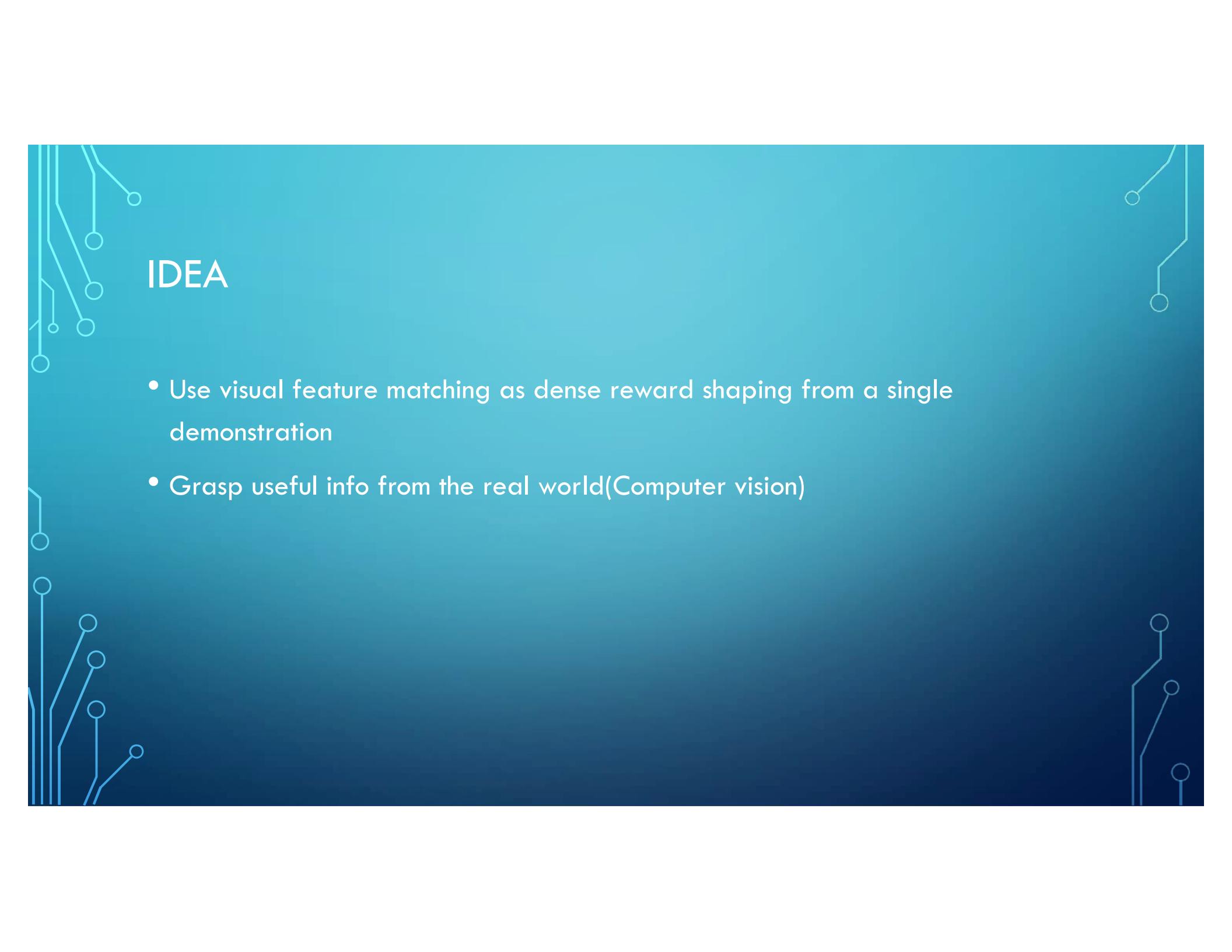
---

## DDPG EXTENSIONS

- Hindsight Experience Replay
- Twin Delayed Deep Deterministic Policy Gradients (TD3)



# VISUAL IMITATION LEARNING



## IDEA

- Use visual feature matching as dense reward shaping from a single demonstration
- Grasp useful info from the real world(Computer vision)

# SINGLE DEMONSTRATION AND MANY INTERACTIONS

- Self-supervised visual learning
- 3D body pose inference through SMPL(a 3D human shape low-parametric model), e.g. keypoint, segmentation and motion reprojections
- PPO & iLQR

## LARGE SCALE DEMONSTRATION COLLECTION

- No/Little interactions
- Challenges: Diversity(Behavioural strategies) & Suboptimality(Lengthier trajectories)
- cVAE(conditional Variational Autoencoder): Generate all possible subgoals(Diversity), Low-level goal-conditioning → unimodal imitation(Suboptimality)

# Review: LQR, iLQR

Recitation 9 10-403

Conor Igoe – May 7 2021

# Review

## LQR Overview

### LQR-Assisted Whole-Body Control of a Wheeled Bipedal Robot with Kinematic Loops

Victor Klemm, Alessandro Morra, Lionel Gulich, Dominik Mannhart,  
David Rohr, Mina Kamel, Yvain de Viragh, and Roland Siegwart

September 2019



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Autonomous Systems Lab

ASC-FNTO

The image shows a screenshot of a YouTube video player. The video itself depicts a black and red wheeled bipedal robot, known as the ANYmal, standing on a light-colored floor against a white backdrop. A wooden stick is being used to apply a force to the robot's side, demonstrating its ability to counteract external disturbances. The video player interface includes the YouTube logo at the top left, a search bar, and various control icons. On the left side of the video frame, there is a vertical black bar. Overlaid text on the video reads "Counteracting external disturbances." At the bottom of the video frame, the caption reads "LQR-Assisted Whole-Body Control of a Wheeled Bipedal Robot with Kinematic Loops (RA-L / ICRA 2020)". Below the video, it says "21,082 views • Feb 11, 2020". The bottom right corner of the player shows a timestamp of "1:28".

# Review

## LQR Overview

- $x, u = s, a ; x \in \mathbb{R}^d, u \in \mathbb{R}^k$
- $x_{t+1} = Ax_t + Bu_t$  Linear dynamics
- $c(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t$  Quadratic costs (can think of as negative of MDP rewards)
- Bellman Equation for deterministic policy  $\pi$  (in cost form):
  - $V^\pi(x) = x^T Q x + \pi(x)^T R \pi(x) + V^\pi(Ax + B\pi(x))$
  - Note: discounting not necessary provided the system is “controllable”
- Bellman Optimality Equation (in cost form):
  - $V^\star(x) = \min_u x^T Q x + u^T R u + V^\star(Ax + Bu)$
- As usual, solving the Bellman Optimality Equation gives us the optimal policy
  - Often this involves solving something known as an **Algebraic Riccati Equation**

# Review

## Algebraic Riccati Equation Overview

$$x_{t+1} = Ax_t + Bu_t$$

$$c(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t$$

- Suppose that  $V^\star$  can be expressed as a quadratic form (this turns out to be true):

- $\bullet V^\star(x) = x^T P^\star x = \min_u x^T Q x + u^T R u + (Ax + Bu)^T P^\star (Ax + Bu) = \min_u f(x, u)$

- Notice that  $f(x, u)$  is convex in  $u$  (this follows from assumptions on  $Q, R$ ), so we can take gradients and set to zero to solve RHS:

- $\bullet \nabla_u f(x, u) = 2Ru + 2B^T P^\star (Ax + Bu) \stackrel{\text{set}}{=} 0 \implies u^\star = -(R + B^T P^\star B)^{-1} B^T P^\star A x = -Kx$

- Plugging back into the Bellman Optimality Equation:

- $\bullet V^\star(x) = x^T P^\star x = \min_u x^T Q x + u^T R u + (Ax + Bu)^T P^\star (Ax + Bu)$

$$= x^T Q x + u^\star^T R u^\star + (Ax + Bu^\star)^T P^\star (Ax + Bu^\star)$$

$$= x^T Q x + x^T K^T R K x + (Ax - BKx)^T P^\star (Ax - BKx)$$

- This yields the following matrix equation (if we solve this then we solve the Bellman Optimality Equation):

- $\bullet P^\star = Q + K^T R K + (A - BK)^T P^\star (A - BK) \quad (\text{where } K = (R + B^T P^\star B^{-1}) B^T P^\star A)$

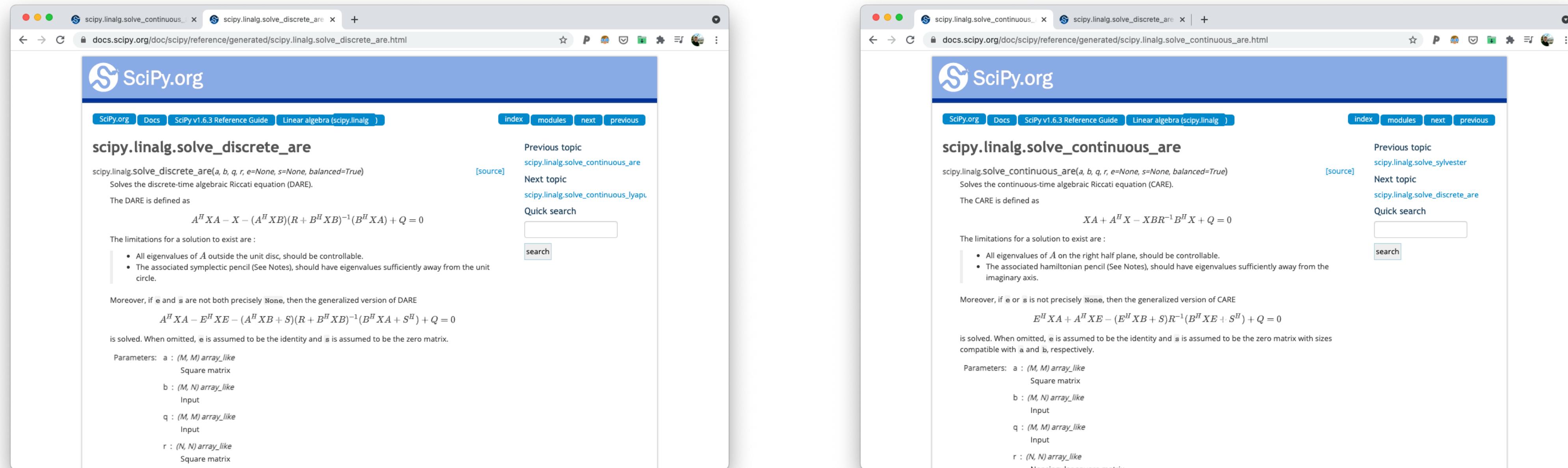
- Substituting  $K$  for the full expression and manipulating we arrive at the **Discrete-time Algebraic Riccati Equation (DARE)**:

- $\bullet P^\star = Q + A^T P^\star A - (A^T P^\star B)(R + B^T P^\star B)^{-1}(B^T P^\star A)$

- Once we solve for  $P^\star$  we can then easily find  $K$  (also known as the gain matrix) to find the optimal policy  $\pi^\star(x) = -Kx$

# Review

## LQR Extensions



- Many variants on the LQR setting, including:
  - Continuous-time LQR (requires solving **CARE** instead of **DARE**)
  - Time-varying LQR with varying dynamics  $A_t, B_t$  (e.g. imagine an ageing motor)
    - Requires a different solution method and yields  $\{K_t\}$  instead of constant  $K$  as in vanilla LQR
  - Having non-zero, time-varying regulation points (vanilla LQR assumes we always wish to keep the state at the origin)
    - If at time  $t$  the regulation point is  $x_t^\star$  then we take action  $-K(x_t - x_t^\star)$  where  $K$  is from vanilla LQR

# Review

## iLQR Overview

- In practice, few systems are truly linear throughout all state-action space
- Popular algorithm: iLQR (similar to Differential Dynamic Programming, DDP). Note that iLQR is a form of Model Predictive Control. Note also that iLQR assumes that the dynamics model is known perfectly (various papers have addressed model misspecification issues and robustness; however we will just focus on the vanilla iLQR algorithm)

Input: initial action sequence  $\{u_0^0, \dots, u_{T-1}^0\}$  and state trajectory  $\{x_0^0, \dots, x_{T-1}^0\}$ , goal state trajectory  $\{x_0^\star, \dots, x_{T-1}^\star\}$

For  $i = 0, 1, \dots, I$ :

1. Linearise  $f$  at each  $(x_t^i, u_t^i)$  yielding  $\{(A_t^i = D_x f(x_t^i, u_t^i), B_t^i = D_u f(x_t^i, u_t^i))\}_{t=0}^{T-1}$  (use finite differences or auto-diff software)
2. Solve a time-varying LQR problem using  $\{(A_t^i, B_t^i)\}$  and  $Q, R$  yielding  $\{K_t^i\}_{t=0}^{T-1}$  (see lecture notes for backward pass equations)
3. Set  $\alpha$  to some positive value and line search over  $\alpha$  until we find an action sequence with better total cost:
  - 3.1. Rollout  $\{\alpha K_t^i\}_{t=0}^{T-1}$  from  $x_0$  using  $\{x_0^\star, \dots, x_{T-1}^\star\}$  to obtain  $\{u_0^{i+1}, \dots, u_{T-1}^{i+1}\}$  and  $\{x_0^{i+1}, \dots, x_{T-1}^{i+1}\}$ :  $u_t^{i+1} = u_t^i - \alpha K_t^i(x_t^{i+1} - x_t^i - x_t^\star)$
  - 3.2.  $\alpha \leftarrow \frac{\alpha}{2}$
4. Break if stopping criteria met (e.g. changes in  $\{u_0^{i+1}, \dots, u_{T-1}^{i+1}\}$  small)

Return  $u_0^I$

# Review

## iLQR Overview

- In practice, few systems are truly linear throughout all state-action space
- Popular algorithm: iLQR (similar to Differential Dynamic Programming, DDP). Note that iLQR is a form of Model Predictive Control. Note also that iLQR assumes that the dynamics model is known perfectly (various papers have addressed model misspecification issues and robustness; however we will just focus on the vanilla iLQR algorithm)

Input: initial action sequence  $\{u_0^0, \dots, u_{T-1}^0\}$  and state trajectory  $\{x_0^0, \dots, x_{T-1}^0\}$ , goal state trajectory  $\{x_0^\star, \dots, x_{T-1}^\star\}$

For  $i = 0, 1, \dots, I$ :

1. Linearise  $f$  at each  $(x_t^i, u_t^i)$  yielding  $\{(A_t^i = D_x f(x_t^i, u_t^i), B_t^i = D_u f(x_t^i, u_t^i))\}_{t=0}^{T-1}$  (use finite differences or auto-diff software)
2. Solve a time-varying LQR problem using  $\{(A_t^i, B_t^i)\}$  and  $Q, R$  yielding  $\{K_t^i\}_{t=0}^{T-1}$  (see lecture notes for backward pass equations)
3. Set  $\alpha$  to some positive value and line search over  $\alpha$  until we find an action sequence with better total cost:
  - 3.1. Rollout  $\{\alpha K_t^i\}_{t=0}^{T-1}$  from  $x_0$  using  $\{x_0^\star, \dots, x_{T-1}^\star\}$  to obtain  $\{u_0^{i+1}, \dots, u_{T-1}^{i+1}\}$  and  $\{x_0^{i+1}, \dots, x_{T-1}^{i+1}\}$ :  $u_t^{i+1} = u_t^i - \alpha K_t^i (x_t^{i+1} - x_t^i - x_t^\star)$
  - 3.2.  $\alpha \leftarrow \frac{\alpha}{2}$
4. Break if stopping criteria met (e.g. changes in  $\{u_0^{i+1}, \dots, u_{T-1}^{i+1}\}$  small)

Return  $u_0^I$

1.  $-x_t^\star$  because not interested in regulating to origin
2.  $-x_t^i$  because LQR solution is w.r.t. linearisation point
3.  $\alpha$  because our linearisation may not hold well and we need to act more closely to the linearisation point
4.  $u_t^i$  because LQR solution is w.r.t. linearisation point

# Review

## iLQR Overview

- Issues with iLQR:
  - Computationally expensive: requires calculating multiple Jacobians and solving multiple LQR problems every time we need to take an action.
  - Mostly suited to “regulator like” problems (i.e. not Atari playing) as ultimately is depending on smoothness in underlying domain to perform well (the “larger” the non-linearity in dynamics the poorer iLQR will perform, although still better than vanilla LQR)

# Review

## iLQR Overview

- Still, lots of robotics problems leverage ideas from iLQR
  - Boston Dynamics recently disclosed Atlas control details, likely builds off iLQR ideas: <https://www.youtube.com/watch?v=EGABAx52GKI>

"Recent Progress on Atlas, the World's Most Dynamic Humanoid Robot" - Scott Kuindersma

9,900 views • Jun 30, 2020

238 likes, 2 dislikes, 1 share, 1 save

RI Seminar: Sidd Srinivasa: Robotic Manipulation... cmurobotics 8.2K views • 4 years ago

- Nonlinear momentum dynamics
  - Iteratively linearize and solve



Boston Dynamics



MBRL

# BASICS

- Intuition: Habitual vs Goal-directed
- Learns an environment/dynamics model/network
- Function Approximation:  $P_\varphi: S \times A \rightarrow R \times S'$
- Model rollout:  $S_t \rightarrow A_t | \pi_\theta(S_t) \rightarrow R_t | P_\varphi(S_t, A_t) \rightarrow S_{t+1} | P_\varphi(S_t, A_t) \rightarrow \dots$

# MODEL-BASED CONTROL

- Model rollout from  $s_0$
- Objective:  $\min_{a_0 \dots a_{T-1}} \|s_{T-1} - s^*\|$  or  $\max_{a_0 \dots a_{T-1}} \sum_{t=0}^{T-1} r_t$

## MPC

- $s_0 \rightarrow a_0 | \pi_\theta(s_0) \rightarrow s_1 | P_\phi(s_0, a_0)$
- Backpropagate using objective  $\max_{a_1 \dots a_{T-1}} \sum_{t=1}^{T-1} r_t$  from model unrolling
- $s_0 \leftarrow s_1$
- Repeat

## ALEATORIC VS EPISTEMIC UNCERTAINTY

- Aleatoric def: Uncertainty due to system/environment(Latin: game of chance)
- Epistemic def: Uncertainty due to insufficient/biased data
- Compounding error(Long term)
- Probabilistic ensemble networks(Gaussian)
- E.g. PETs & MBPO

---

**Algorithm 2** Model-Based Policy Optimization with Deep Reinforcement Learning

---

- 1: Initialize policy  $\pi_\phi$ , predictive model  $p_\theta$ , environment dataset  $\mathcal{D}_{\text{env}}$ , model dataset  $\mathcal{D}_{\text{model}}$
  - 2: **for**  $N$  epochs **do**
  - 3:   Train model  $p_\theta$  on  $\mathcal{D}_{\text{env}}$  via maximum likelihood
  - 4:   **for**  $E$  steps **do**
  - 5:     Take action in environment according to  $\pi_\phi$ ; add to  $\mathcal{D}_{\text{env}}$
  - 6:     **for**  $M$  model rollouts **do**
  - 7:       Sample  $s_t$  uniformly from  $\mathcal{D}_{\text{env}}$
  - 8:       Perform  $k$ -step model rollout starting from  $s_t$  using policy  $\pi_\phi$ ; add to  $\mathcal{D}_{\text{model}}$
  - 9:     **for**  $G$  gradient updates **do**
  - 10:      Update policy parameters on model data:  $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$
-

# OFF-POLICY RL

## DEFINITION

- Learns value of the optimal policy disregarding agent's actions
- Behavior policy(Correlated with current)
- On policy algos: SARSA, policy gradient methods
- Off policy algos: Q-learning, soft a2c, DDPG

## DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

- **Objective:**  $\max_{\theta} \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [\sum_{t=0}^{T-1} Q(s_t, a_t)]$
- $\nabla_{\theta} \mathbb{E}[\sum_{t=0}^{T-1} Q(s_t, a_t)] = \mathbb{E}[\sum_{t=0}^{T-1} \nabla_{a_t} Q(s_t, a_t) \nabla_{\theta} a_t]$
- Learns from experience tuples in a buffer(Not fixed)
- Extrapolation error: Distinguish good (s,a)s from the bad ones(states visited by policy).
- Not truly off-policy

---

**Algorithm 1** Deep Deterministic Policy Gradient

---

```

1: Input: initial policy parameters  $\theta$ , Q-function parameters  $\phi$ , empty replay buffer  $\mathcal{D}$ 
2: Set target parameters equal to main parameters  $\theta_{\text{targ}} \leftarrow \theta$ ,  $\phi_{\text{targ}} \leftarrow \phi$ 
3: repeat
4:   Observe state  $s$  and select action  $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$ , where  $\epsilon \sim \mathcal{N}$ 
5:   Execute  $a$  in the environment
6:   Observe next state  $s'$ , reward  $r$ , and done signal  $d$  to indicate whether  $s'$  is terminal
7:   Store  $(s, a, r, s', d)$  in replay buffer  $\mathcal{D}$ 
8:   If  $s'$  is terminal, reset environment state.
9:   if it's time to update then
10:    for however many updates do
11:      Randomly sample a batch of transitions,  $B = \{(s, a, r, s', d)\}$  from  $\mathcal{D}$ 
12:      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

13:      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s, a, r, s', d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$$

14:      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

15:      Update target networks with

$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho \phi_{\text{targ}} + (1 - \rho) \phi \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho) \theta \end{aligned}$$

16:    end for
17:  end if
18: until convergence

```

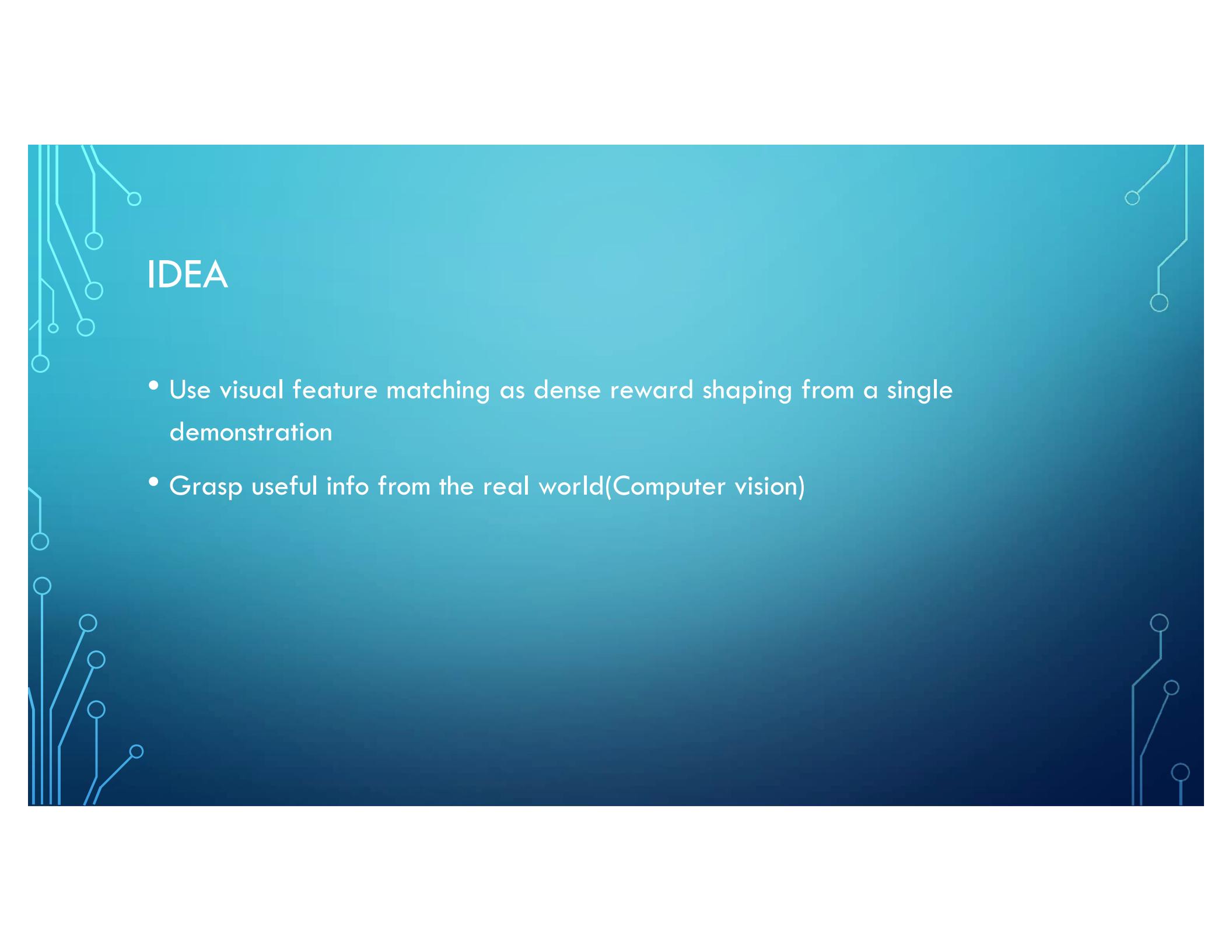
---

## DDPG EXTENSIONS

- Hindsight Experience Replay
- Twin Delayed Deep Deterministic Policy Gradients (TD3)



# VISUAL IMITATION LEARNING



## IDEA

- Use visual feature matching as dense reward shaping from a single demonstration
- Grasp useful info from the real world(Computer vision)

# SINGLE DEMONSTRATION AND MANY INTERACTIONS

- Self-supervised visual learning
- 3D body pose inference through SMPL(a 3D human shape low-parametric model), e.g. keypoint, segmentation and motion reprojections
- PPO & iLQR

## LARGE SCALE DEMONSTRATION COLLECTION

- No/Little interactions
- Challenges: Diversity(Behavioural strategies) & Suboptimality(Lengthier trajectories)
- cVAE(conditional Variational Autoencoder): Generate all possible subgoals(Diversity), Low-level goal-conditioning → unimodal imitation(Suboptimality)

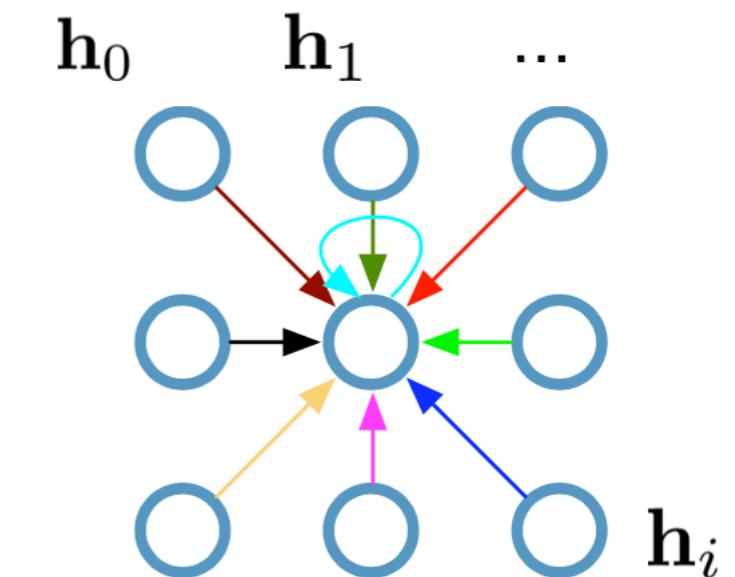
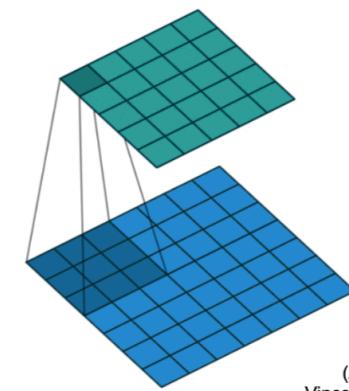
# **Graph Neural Network / Learning from Demonstrations and Task Rewards / Adversarial Imitation Learning**

**A review**

---

# **Graph Neural Network**

### Single CNN layer with 3x3 filter:



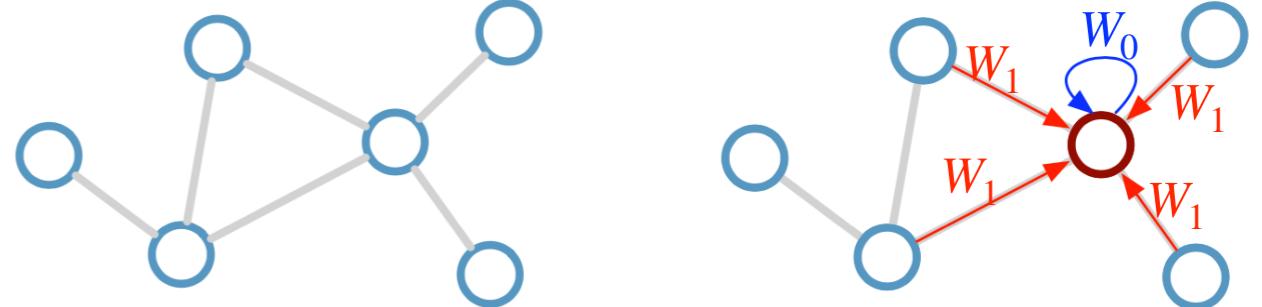
## From CNN to GNN

Hidden layer activations of pixels/nodes  $h_i \in \mathbb{R}^F$

$$h_4^{(l+1)} = \sigma \left( \underbrace{W_0^{(l)} h_0^{(l)}}_{\text{---}} + \underbrace{W_1^{(l)} h_1^{(l)}}_{\text{---}} + \dots + \underbrace{W_8^{(l)} h_8^{(l)}}_{\text{---}} \right)$$

• Different weight!

Undirected graph



## From CNN to GNN

Hidden layer activations of nodes  $h_i \in \mathbb{R}^F$

Neighbor indices  $\mathcal{N}_i$

Fixed, trainable norm  $c_{ij}$

$$h_i^{(l+1)} = \sigma \left( W_0^{(l)} h_i^{(l)} + \sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} W_1^{(l)} h_j^{(l)} \right)$$

- Sharing weight over neighboring locations
- Permutation invariant
- Linear complexity

## Paper I

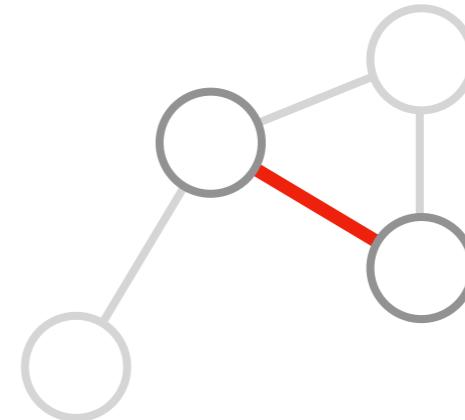
*Interaction Networks for Learning about Objects, Relations and Physics, Battaglia et al., 2016*

- Object / object part → graph's node
- Input
  - Object state
    - Dynamic: position, velocity
    - Static: mass, size, shape
  - Relation attribute
    - Coefficients of connectivity, restitution, spring constant, etc.
- Output: future velocity

## Paper I

*Interaction Networks for Learning about Objects, Relations and Physics, Battaglia et al., 2016*

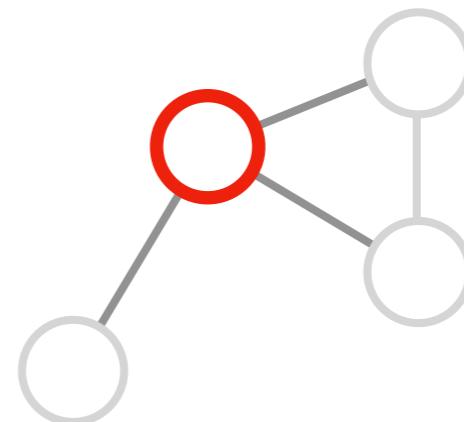
- Relational / **edge** NN
  - Input
    - Two object (two sides of an edge) states
    - Relational attributes
  - Output
    - Feature vector



## Paper I

*Interaction Networks for Learning about Objects, Relations and Physics, Battaglia et al., 2016*

- Object / node NN
  - Input
    - One object state
    - Summation of all incoming edge message
  - Output
    - Future object velocity



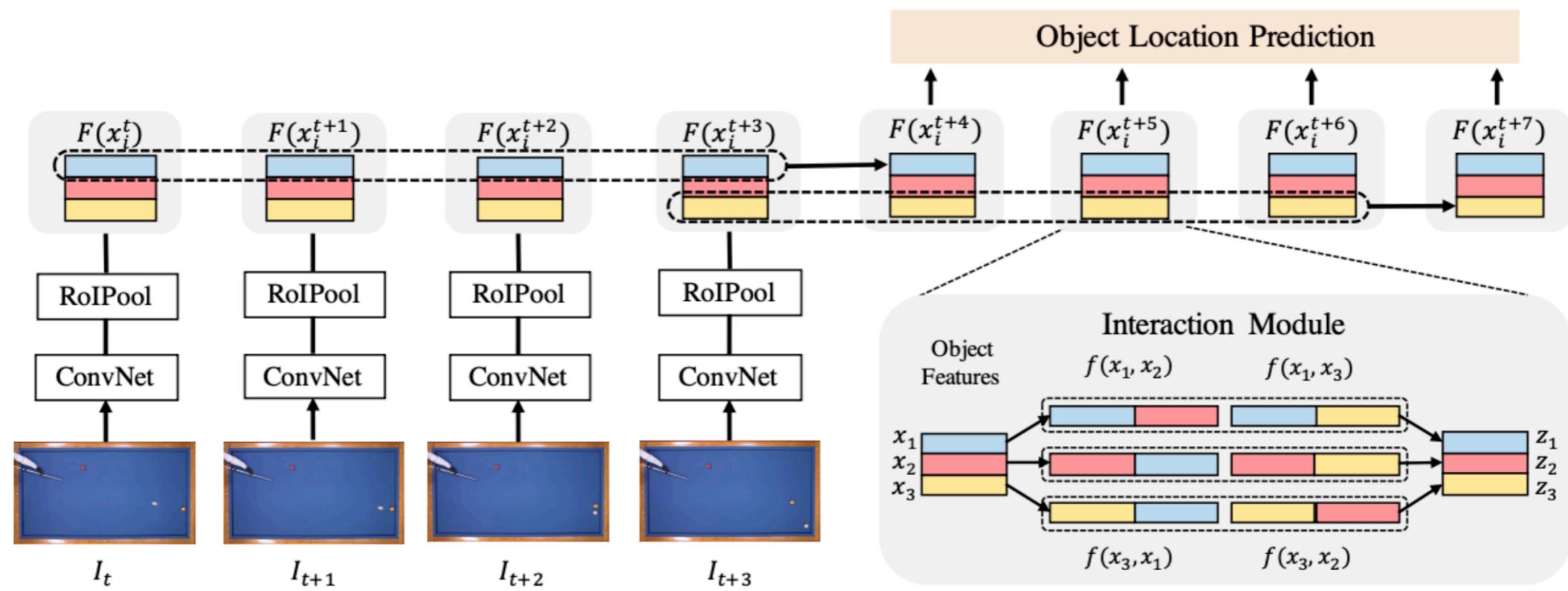
## Paper II

*Long-term Visual Dynamics with Region Proposal Interaction Networks, Qi et al., 2021*

- Learning object interactions from visual input
- Input
  - Detected object appearance (ConvNet, pooling)
  - Motion history
- Output
  - Future motion trajectory

## Paper II

*Long-term Visual Dynamics with Region Proposal Interaction Networks, Qi et al., 2021*



## Paper III

*Graph Networks as Learnable Physics Engines for Inference and Control,  
Gonzalez et al., 2018*

- Physical system's bodies and joints → graph's nodes and edges
- Input
  - Global feature
    - None / Gravity, wind, density, etc.
  - Node feature
    - Dynamic: position, 4D quaternion orientation, linear and angular velocity
    - Static: mass, inertia, etc.
  - Edge feature
    - Magnitude of action at joint, etc.
- Output: dynamic features, temporal difference.

## Paper III

*Graph Networks as Learnable Physics Engines for Inference and Control,  
Gonzalez et al., 2018*

### G.1. Dynamic graph

We retrieved the the absolute position, orientation, linear and angular velocities for each body:

- **Global: None**
- **Nodes:** (for each body)
  - Absolute body position (3 vars): `mjData.xpos`
  - Absolute body quaternion orientation position (4 vars): `mjData.xquat`
  - Absolute linear and angular velocity (6 vars): `mj_objectVelocity (mjOBJ_XBODY, flg_local=False)`
- **Edges:** (for each joint) Magnitude of action at joint: `mjData.ctrl` (0, if not applicable).

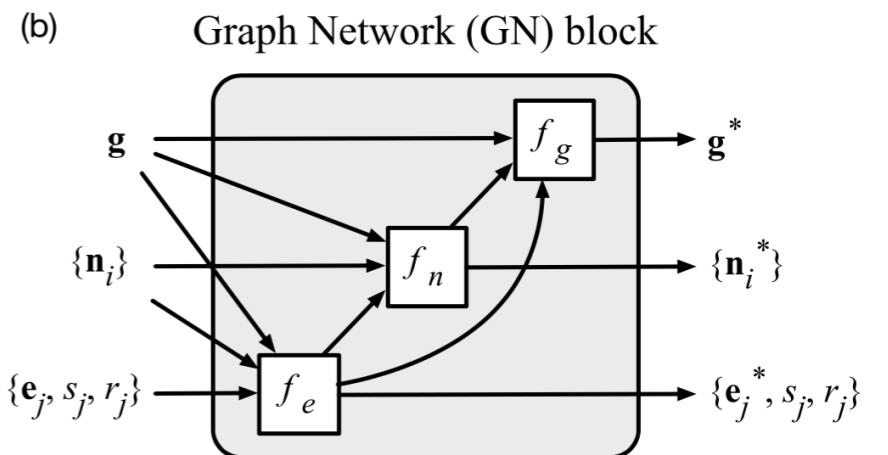
## Paper III

*Graph Networks as Learnable Physics Engines for Inference and Control,  
Gonzalez et al., 2018*

### G.2. Static graph

We performed an exhaustive selection of global, body, and joint static properties from mjModel:

- **Global:** mjModel.opt.{timestep, gravity, wind, magnetic, density, viscosity, impratio, o\_margin, o\_solref, o\_solimp, collision\_type (one-hot), enableflags (bit array), disableflags (bit array)}.
- **Nodes:** (for each body) mjModel.body\_{mass, pos, quat, inertia, ipos, iquat}.
- **Edges:** (for each joint)  
Direction of edge (1: parent-to-child, -1: child-to-parent).  
Motorized flag (1: if motorized, 0 otherwise).  
Joint properties: mjModel.jnt\_{type (one-hot), axis, pos, solimp, solref, stiffness, limited, range, margin}.  
Actuator properties: mjModel.opt.actuator\_{biastype (one-hot), biasprm, cranklength, ctrllimited, ctrlrange, dyntype (one-hot), dynprm, forcelimited, forcerange, gaintype (one-hot), gainprm, gear, invweight0, length0, lengthrange}.



## Paper III

*Graph Networks as Learnable Physics Engines for Inference and Control,  
Gonzalez et al., 2018*

---

### Algorithm 1 Graph network, GN

---

**Input:** Graph,  $G = (\mathbf{g}, \{\mathbf{n}_i\}, \{\mathbf{e}_j, s_j, r_j\})$   
**for** each edge  $\{\mathbf{e}_j, s_j, r_j\}$  **do**  
    Gather sender and receiver nodes  $\mathbf{n}_{s_j}, \mathbf{n}_{r_j}$   
    Compute output edges,  $\mathbf{e}_j^* = f_e(\mathbf{g}, \mathbf{n}_{s_j}, \mathbf{n}_{r_j}, \mathbf{e}_j)$   
**end for**  
**for** each node  $\{\mathbf{n}_i\}$  **do**  
    Aggregate  $\mathbf{e}_j^*$  per receiver,  $\hat{\mathbf{e}}_i = \sum_{j/r_j=i} \mathbf{e}_j^*$   
    Compute node-wise features,  $\mathbf{n}_i^* = f_n(\mathbf{g}, \mathbf{n}_i, \hat{\mathbf{e}}_i)$   
**end for**  
    Aggregate all edges and nodes  $\hat{\mathbf{e}} = \sum_j \mathbf{e}_j^*$ ,  $\hat{\mathbf{n}} = \sum_i \mathbf{n}_i^*$   
    Compute global features,  $\mathbf{g}^* = f_g(\mathbf{g}, \hat{\mathbf{n}}, \hat{\mathbf{e}})$   
**Output:** Graph,  $G^* = (\mathbf{g}^*, \{\mathbf{n}_i^*\}, \{\mathbf{e}_j^*, s_j, r_j\})$

---

- 
- Relation between node, edge, global features
  - Sequence of update
  - Architecture of GN block

## Paper IV

*Learning to Simulate Complex Physics with Graph Networks, Gonzalez et al., 2020*

- Object → graph of particles, scene → graph of all particles from all objects
- Input
  - Particle velocity of last 5 time steps
- Output: particle acceleration

## Paper IV

*Learning to Simulate Complex Physics with Graph Networks, Gonzalez et al., 2020*

- Particles are different from nodes. Only consider displacements.
- Inject noise to particle velocities during training time to tackle accumulating error.
- Use edges to encode two particles' relative position. No longer use absolute position compared to previous papers' node features. This is for translation invariance.
- Use multiple rounds of message passing, each round has different node, edge weights.

---

# **Learning from Demonstrations and Task Rewards**

## **Learning from demonstrations**

- Accelerate trial-and-error learning by suggesting good actions to try
- Train initial safe policies, to deploy in the real world
- Time consuming
- Sub-optimality, noise, diversity in performing task
- Can not surpass expert

## **Learning from task rewards**

- Cheap supervision
- Right end task encoded via task rewards
- Sample inefficient
- Unsafe initial policy, unsafe to deploy in real world

## **Learning from demonstrations and task rewards**

### **Major considerations**

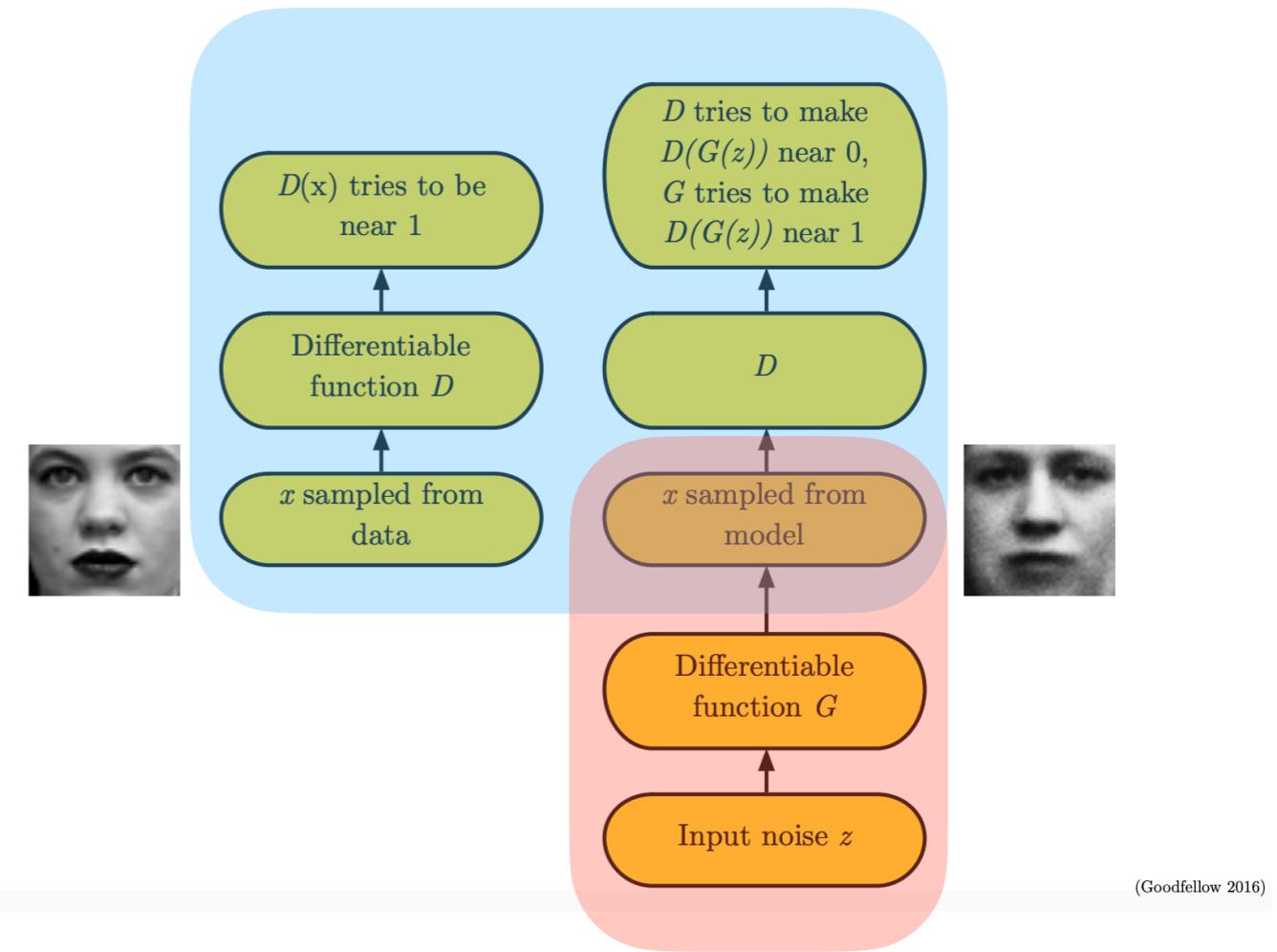
- Sample efficiency
- Outperforming expert
- Safe to deploy in real world with good/safe initial performance

## **Learning from demonstrations and task rewards**

- Initialize the replay buffer with demonstrations
- Pre-train the model-free RL method with a demonstration only buffer, then fine-tune it.
- Combine imitation and task rewards (**see in GAIL**)
- Exploit the temporal structure, and solve longer horizon tasks progressively, rather than solving at once.

---

# **Adversarial Imitation Learning**



## Generative Adversarial Nets (GAN)

Discriminator  $D$

Generator  $G$

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

- Please refer to lecture slides for explanation of min-max game, which is very clear and specific.
- We usually use Adam (adaptive optimizer) to train discriminator and generator simultaneously.
- It is important to maintain a *balance* between the discriminator and generator, i.e. avoiding one being much stronger than the other.

(Goodfellow 2016)

## Generative Adversarial Imitation Learning (GAIL)

Discriminator  $D_\phi$

- Distinguish between state-action pairs from expert demonstrations and ones generated/visited by agent policy  $\pi_\theta$

Generator  $\pi_\theta$

- Policy network that conditions on state  $\pi_\theta(s)$

Reward  $r(s, a) = \log D_\phi(s, a), (s, a) \sim \pi_\theta$

- “*The reward for the policy optimization is how well I matched the demonstrator’s trajectory distribution, else, how well I confused the discriminator.*”

## Generative Adversarial Imitation Learning (GAIL)

How to update the discriminator  $D_\phi$  ?

$$\mathbb{E}_{(s,a) \sim Demo} \left[ \nabla_\phi \log(1 - D_\phi(s, a)) \right] + \mathbb{E}_{(s,a) \in \tau_i} \left[ \nabla_\phi \log D_\phi(s, a) \right]$$

How to update the generator / policy  $\pi_\theta$  ? Policy gradient (e.g. REINFORCE)

$$\mathbb{E}_{(s,a) \in \tau_i} \left[ \nabla_\theta \log \pi_\theta \log D_{\phi_{i+1}}(s, a) \right]$$

## **GAIL vs. BC (Behavior Cloning)**

**GAIL outperforms BC, why?**

- GAIL has more interaction with environment
- BC reduces imitation learning to supervised learning w.r.t. single action.

## GAIL vs. BC (Behavior Cloning)

### 7 Discussion and outlook

As we demonstrated, our method is generally quite sample efficient in terms of expert data. However, it is not particularly sample efficient in terms of environment interaction during training. The number of such samples required to estimate the imitation objective gradient (18) was comparable to the number needed for TRPO to train the expert policies from reinforcement signals. We believe that we could significantly improve learning speed for our algorithm by initializing policy parameters with behavioral cloning, which requires no environment interaction at all.

Fundamentally, our method is model free, so it will generally need more environment interaction than model-based methods. Guided cost learning [7], for instance, builds upon guided policy search [13] and inherits its sample efficiency, but also inherits its requirement that the model is well-approximated by iteratively fitted time-varying linear dynamics. Interestingly, both our Algorithm 1 and guided cost learning alternate between policy optimization steps and cost fitting (which we called discriminator fitting), even though the two algorithms are derived completely differently.

Our approach builds upon a vast line of work on IRL [31, 1, 29, 28], and hence, just like IRL, our approach does not interact with the expert during training. Our method explores randomly to determine which actions bring a policy’s occupancy measure closer to the expert’s, whereas methods that do interact with the expert, like DAgger [24], can simply ask the expert for such actions. Ultimately, we believe that a method that combines well-chosen environment models with expert interaction will win in terms of sample complexity of both expert data and environment interaction.

## **GAIL combining imitation and task rewards**

**Original reward**

$$r(s, a) = \log D_\phi(s, a), (s, a) \sim \pi_\theta$$

**Combined reward**

$$r(s, a) = \lambda r_{GAIL}(s, a) + (1 - \lambda)r_{task}(s, a), \lambda \in [0, 1]$$

$$r_{GAIL}(s, a) = -\log(1 - D(s, a))$$

## Paper

*Reinforcement and Imitation Learning for Diverse Visuomotor Skills, Zhu et al., 2018*

- Combines imitation and task rewards
- Start episodes by setting the world in states of the demonstration trajectories.
- Please review the paper.

# **Thank you**

---

**A review**

**Yuning Wu, 5/7/2021**