Quiz 3 Review

Part 1

Part 1 of Quiz 3 Review

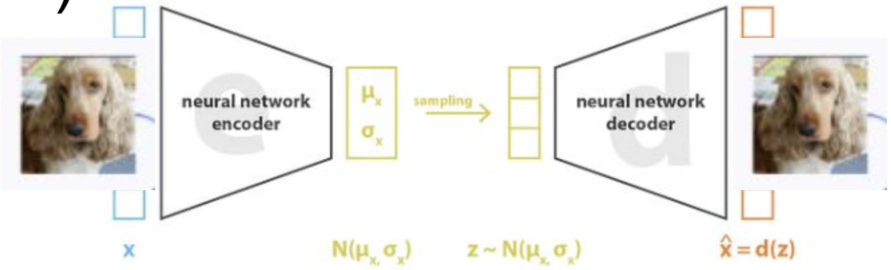
Lectures #18-22

- A little more MBRL
- Ideas for Intelligent Exploration
- Offline RL
- Sim2Real Transfer

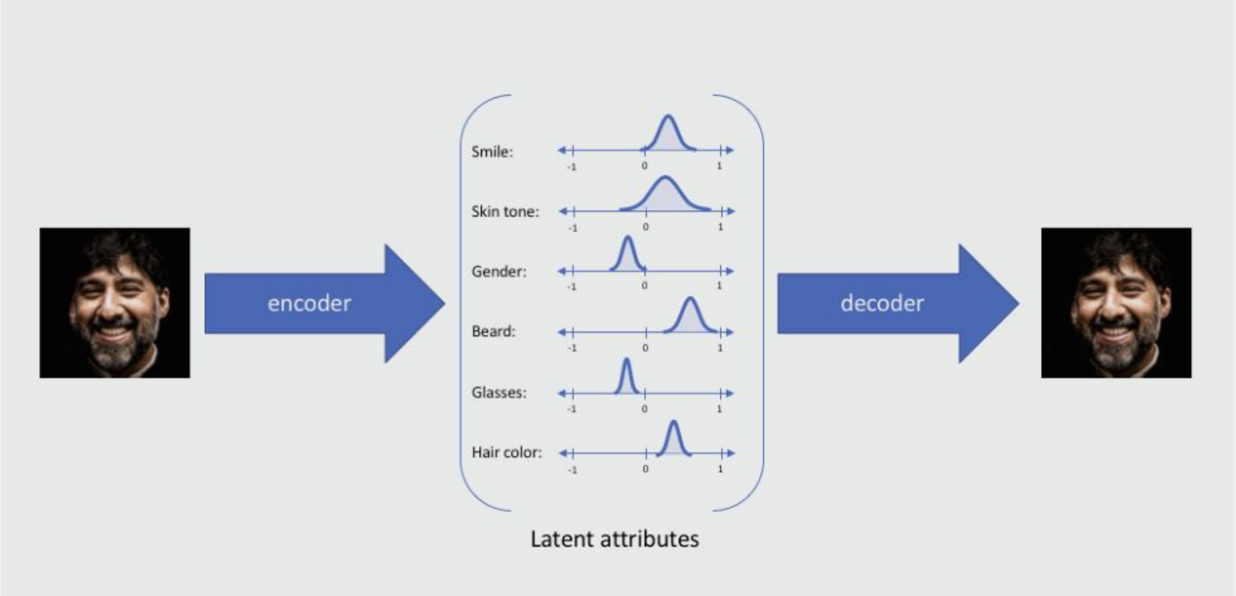
⇒ Part 2 next week will cover the rest of the scope

Some more MBRL

Variational Autoencoders (VAE)



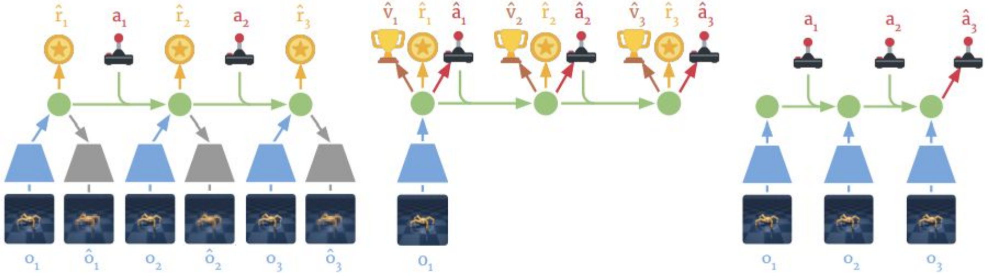
Can also condition the decoder on other variables (conditional VAE)



Check out

- Slides for loss derivation
- "[Tutorial on Variational Autoencoders]"

Dreamer



(a) Learn dynamics from experience (b) Learn behavior in imagination (c) Act in the environment

Algorithm 1: Dreamer

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize neural network parameters θ, ϕ, ψ randomly.
while *not converged* **do**
  for *update step* $c = 1..C$ **do**
    `// Dynamics learning`
    Draw B data sequences $\{(a_t, o_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}$.
    Compute model states $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$.
    Update θ using representation learning.
    `// Behavior learning`
    Imagine trajectories $\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$ from each s_t.
    Predict rewards $\mathrm{E}\big(q_\theta(r_\tau \mid s_\tau)\big)$ and values $v_\psi(s_\tau)$.
    Compute value estimates $\mathrm{V}_\lambda(s_\tau)$ via Equation 6.
    Update $\phi \leftarrow \phi + \alpha \nabla_\phi \sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau)$.
    Update $\psi \leftarrow \psi - \alpha \nabla_\psi \sum_{\tau=t}^{t+H} \frac{1}{2} \big\| v_\psi(s_\tau) - \mathrm{V}_\lambda(s_\tau) \big\|^2$.
  `// Environment interaction`
  $o_1 \leftarrow$ `env.reset()`
  for *time step* $t = 1..T$ **do**
    Compute $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$ from history.
    Compute $a_t \sim q_\phi(a_t \mid s_t)$ with the action model.
    Add exploration noise to action.
    $r_t, o_{t+1} \leftarrow$ `env.step(`a_t`)`.
  Add experience to dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^T\}$.

Model components

Representation	$p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$
Transition	$q_\theta(s_t \mid s_{t-1}, a_{t-1})$
Reward	$q_\theta(r_t \mid s_t)$
Action	$q_\phi(a_t \mid s_t)$
Value	$v_\psi(s_t)$

Hyper parameters

Seed episodes	S
Collect interval	C
Batch size	B
Sequence length	L
Imagination horizon	H
Learning rate	α

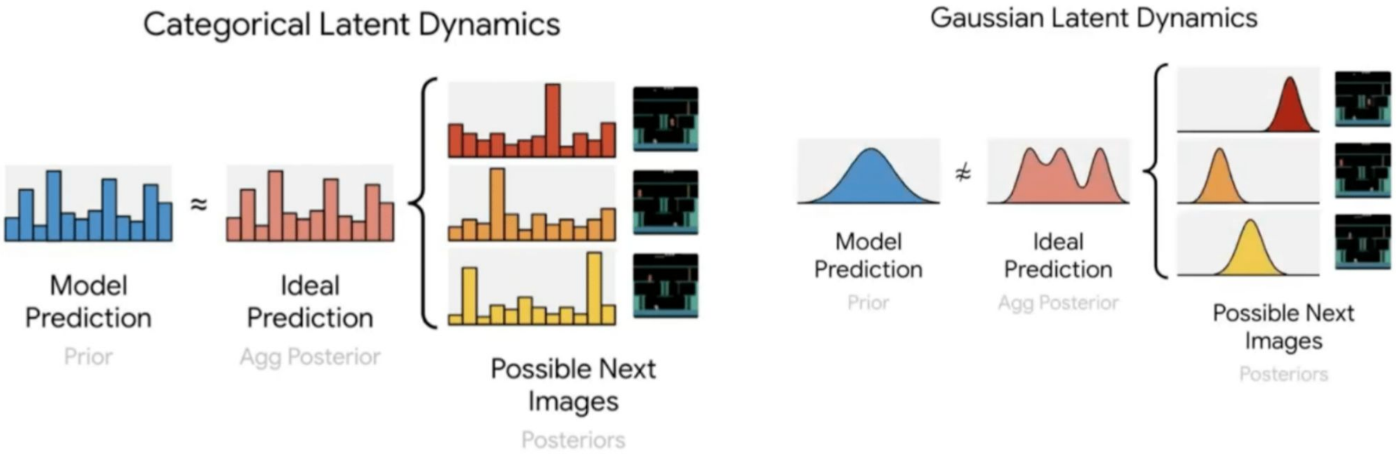
Learn a model of the environment (predict next state)

Train on imagined trajectories!

Act in the environment to get more observations for step 1

Discrete variables better capture multi-modal distributions

DREAMER v2



Intelligent Exploration

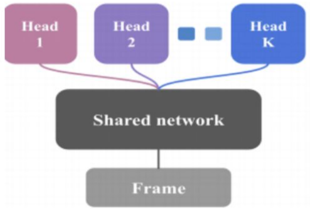
Curiosity-driven exploration

- Ensembles of Q functions: modeling uncertainty of Q values
- State counting: the lower the count of the state the higher the exploration bonus
- Model prediction error: the higher the prediction error the higher the curiosity
- Reachability: the least reachable a state from a set of already reached states in my memory, the higher the exploration bonus
- Non-parametric memory of states and their transitions (reachability) of one to the other. Explore by maximizing coverage.

Exploration via modeling uncertainty of Q function

Model distribution itself (difficult)

1. **Bayesian neural networks**. Estimate posteriors for the neural weights, as opposed to point estimates. We just saw that..
2. **Neural network ensembles.** Train multiple Q-function approximations each on using different subset of the data. A reasonable approximation to 1.
3. **Neural network ensembles with shared backbone.** Only the heads are trained with different subset of the data. A reasonable approximation to 2 with less computation.



4. **Ensembling by dropout.** Randomly mask-out (zero out)neural network weights, to create different neural nets, both at train and test time. reasonable approximation to 2.

Exploration via modeling uncertainty of Q function

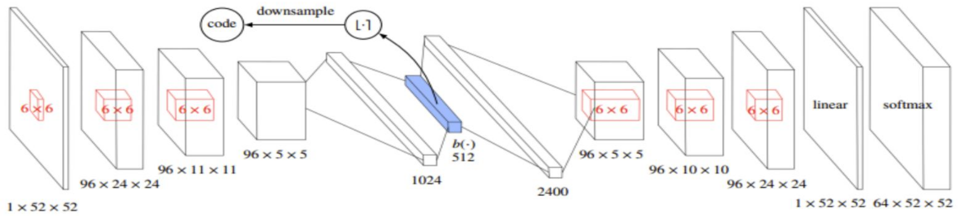
With ensembles we achieve similar things as with Bayesian nets:

- The entropy of predictions of the network (obtained by sampling different heads) is high in the no data regime. Thus, Q function values will have high entropy there and encourage exploration.
- When Q values have low entropy, i exploit, i do not explore.

State counting

State Counting with DeepHashing

- We count states (images) but not in pixel space, but in latent compressed space.
- Compress s into a latent code, then count occurrences of the code.
- How do we get the image encoding? E.g, using autoencoders.



- Note: There is no guarantee such reconstruction loss will capture the important things that make two states to be similar or not policy wise..

Map a state to a hash code, then count up states visited with that hash code. Encourage visiting states with low count hash codes

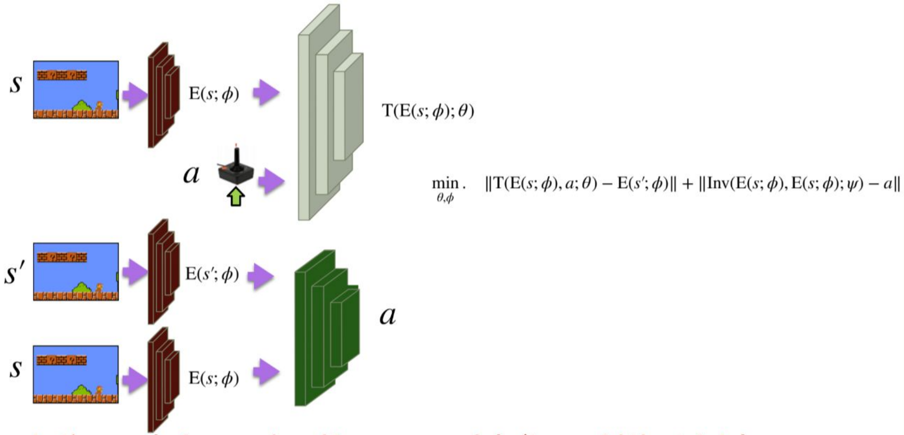
$$R^t(s,a,s') = \underbrace{r(s,a,s')}_{\text{extrinsic}} + \underbrace{\mathscr{B}^t(\phi(s))}_{\text{intrinsic}}$$

Exploration A Study of Count-Based Exploration for Deep Reinforcement Learning,Tang et al.

Prediction error

Learning Visual Dynamics

Exploration reward bonus $\mathscr{B}^t(s, a, s') = \|\mathrm{T}(\mathrm{E}(s;\phi), a; \theta) - \mathrm{E}(s';\phi)\|$



- Let's couple forward and inverse models (to avoid the trivial solution)
- ...then we will only predict things that the agent can control

Curiosity driven exploration with self-supervised prediction, *Pathak*

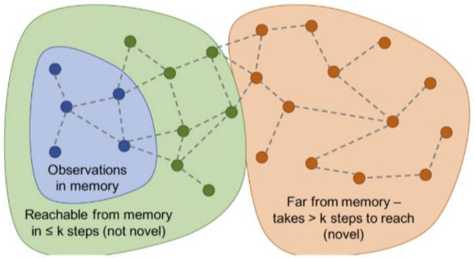
Limitation of Prediction Error as Bonus

- Agent will be rewarded even though the model cannot improve.
- The agent is attracted forever in the most noisy states, with unpredictable outcomes.
- If we give the agent a TV and a remote, it becomes a couch potato!

Curiosity driven exploration with self-supervised prediction, Pathak et al.
Large-scale study of Curiosity-Driven Learning, Burda et al.

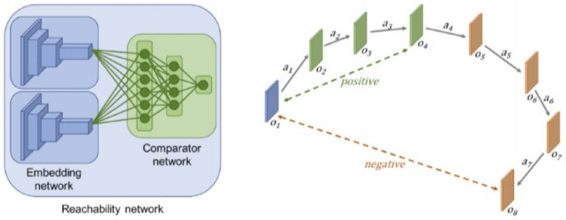
Reachability - episodic curiosity through reachability

non-parametric memory structure



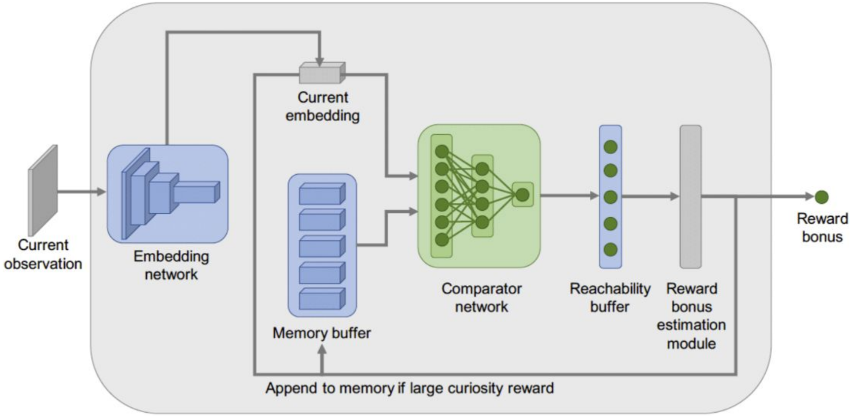
- We will be using augmented rewards as before $R^t(s,a,s') = \underbrace{r(s,a,s')}_{extrinsic} + \underbrace{\mathscr{B}^t(s,\mathscr{M})}_{intrinsic}$, where $\mathscr{M}$ is a non-parametric memory structure populated with embeddings of past image observations.
- Curiosity reward will use a comparator neural net, that takes as input two images and predicts whether they are close (few actions apart) or far
- We will plug those rewards into PPO, a model-free RL method

comparator network trained with temporal contrastive learning



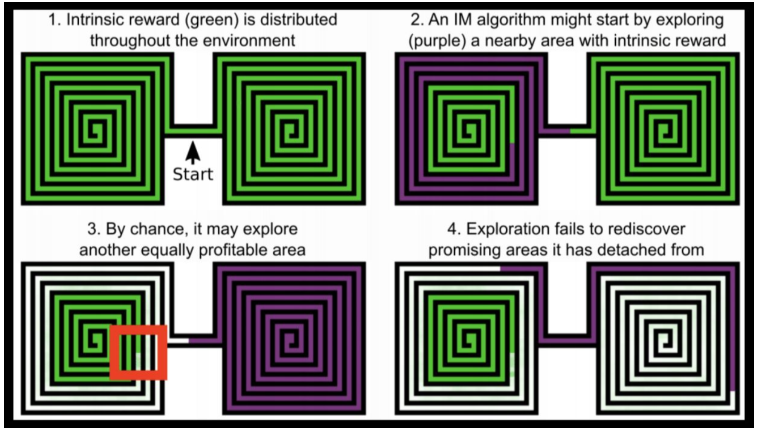
$$\mathscr{L}_\phi(o,o^+,o^-) = \|E(o,\phi) - E(o^+,\phi)\| + \max(0, \gamma - \|E(o,\phi) - E(o^-,\phi)\|)$$

At each time step the agent compares the current observation with the ones in memory. If it is novel (takes more steps to reach than a threshold) then agent get rewarded, and the novel observation is added into memory.



Go-Explore: a New Approach for Hard-Exploration Problems

Failures of intrinsic motivation stem from two issues:



Detachment is the idea that an agent driven by intrinsic motivation could become detached from the frontiers of high intrinsic reward (IR).

a. Once IR is obtained, the agent will not remember how to get back to that location (catastrophic forgetting)
b. The Go-Explore algorithm addresses detachment by explicitly storing an archive of promising states visited so that they can then be revisited and explored from later.

Derailment can occur when an agent has discovered a promising state and it would be beneficial to return to that state and explore from it.

a. IR causes agents to not want to return to those states to explore from there
b. To address derailment, an insight in Go-Explore is that effective exploration can be decomposed into first returning to a promising state (without intentionally adding any exploration) before then exploring further.

Go-explore

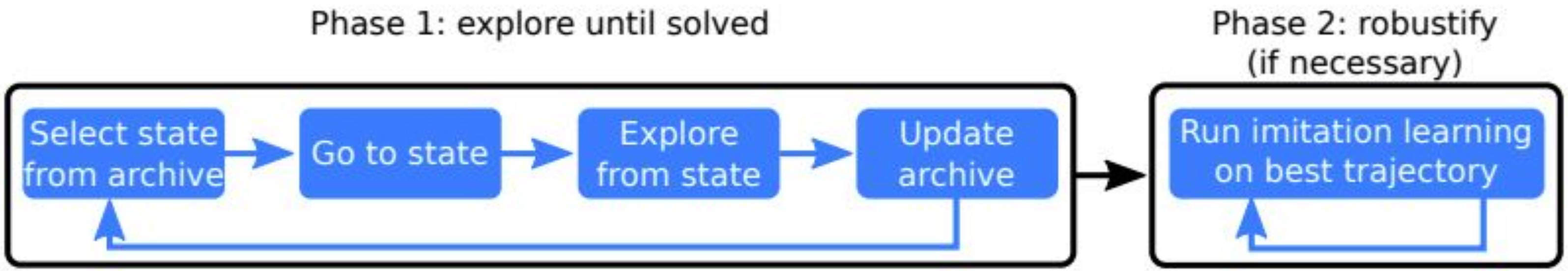


Figure 2: **A high-level overview of the Go-Explore algorithm.**

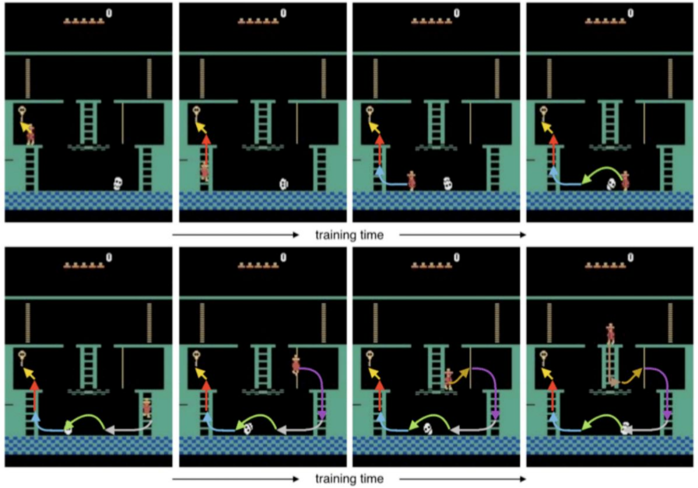
1. Phase 1
   a. (deterministic) Go to state in archive, then explore randomly, update archive with shortest path to that state -> replace existing if path got higher score or shorter path with same score
   b. Sparsify states by downsampling image and use this for determining “same states”
2. Phase 2
   a. Run IL on best trajectories from phase 1 to make policy more “robust”

Learning Montezuma's Revenge from a Single Demonstration

- RL is very sample inefficient especially in sparse reward settings (may never reach the reward)

- IL also requires many demos to do well

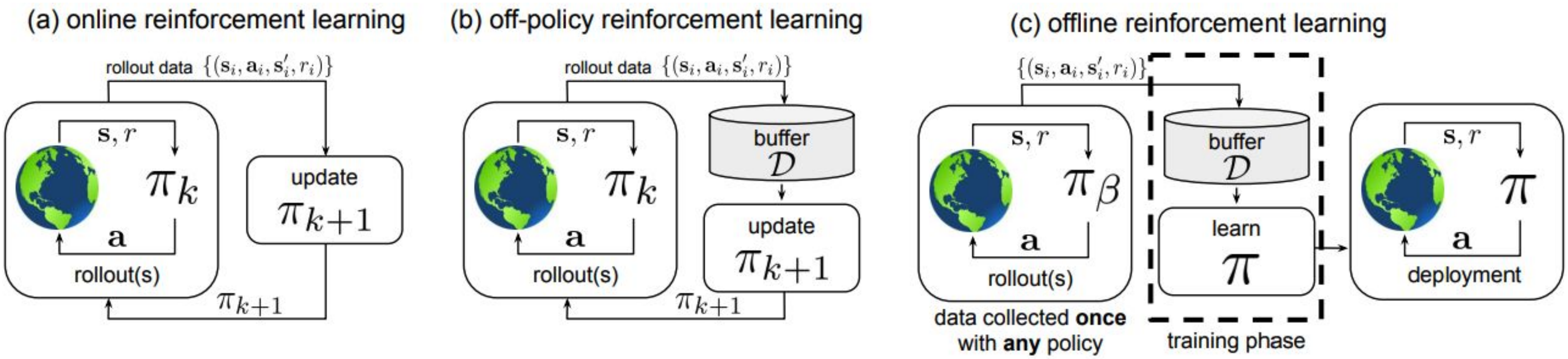
- This paper: learn from single demo in sparse reward setting by backtracking a small amount from the reward. Do this iteratively until at starting state.



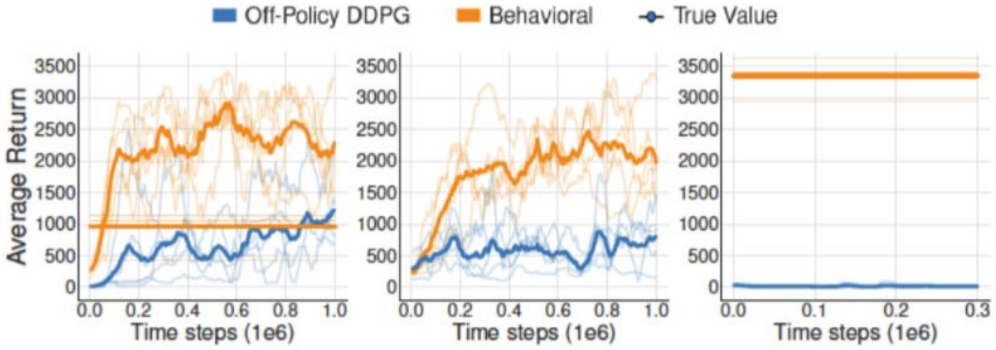
Offline RL

Offline RL Setting

a.k.a *batch* RL (fixed batch of data to train policy with)



Offline RL Setting - naive off-policy methods do not work



(a) Final buffer performance (b) Concurrent performance (c) Imitation performance

$$Q(s,a) \leftarrow r + \gamma Q(s',a')$$

GIVEN GENERATED

Off-policy DDPG doesnt learn good behaviors

The Difference?

1. Agent orange: Interacted with the environment.
   - Standard RL loop.
   - Collect data, store data in buffer, train, repeat.

2. Agent blue: Never interacted with the environment.
   - Trained with data collected by agent orange concurrently.

Why model-free RL does not work with fixed experience buffers?

Extrapolation error:

The Q-function trained from a fixed experience buffer has no way of knowing whether the actions not contained in the buffer are better or worse.

One Solution: Batch Constrained Q-learning (BCQ)

BCQ learns a policy with a similar state-action visitation to the data in the batch

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a)+\alpha(r+\gamma \max_{a' \text{s.t.} (s',a')\in\mathcal{B}} Q(s',a')).$$

Train a generative model to provide action samples that match the action samples in the batch:

$$\pi(s) = \operatorname*{argmax}_{a_i+\xi_\phi(s,a_i,\Phi)} Q_\theta(s, a_i + \xi_\phi(s,a_i,\Phi)),$$
$$\{a_i \sim \boxed{G_\omega(s)}\}_{i=1}^n.$$

A state conditioned generative model that predicts actions given a state that are contained in the batch B

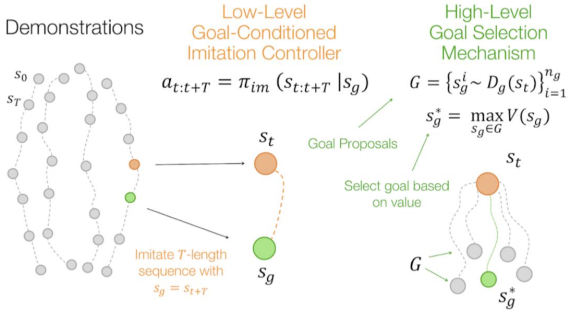
IRIS: Implicit Reinforcement without Interaction at Scale for Learning Control from Offline Robot Manipulation Data

Challenges from Large Scale Demo Datasets:

- Diversity (each behavior has diverse solutions)
- Suboptimality (make mistakes, etc.)

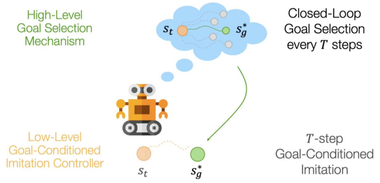
Decompose into subgoals

- The IRIS algorithm uses a high-level mechanism and a low-level controller to make decisions.
- The high-level mechanism selects a new goal state that is held constant for the next T timesteps, while the low-level controller is conditioned on this goal state to try and reach it.
- cVAE generates set of goal proposals, value function evaluates them - select goal with highest value



- A cVAE that generates all possible subgoal states reachable within T steps from s_t
- A task specific value function that scores subgoals trained with batch-constrained Q learning.

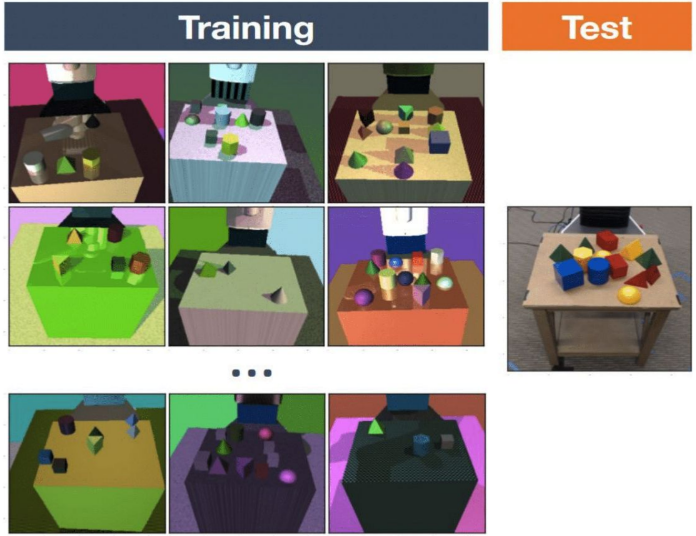
Acting at test time



A subgoal is proposed+selected every T timesteps, that the low level policy tries to achieve. Repeat.

Sim2Real Transfer

Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World, Tobin et al

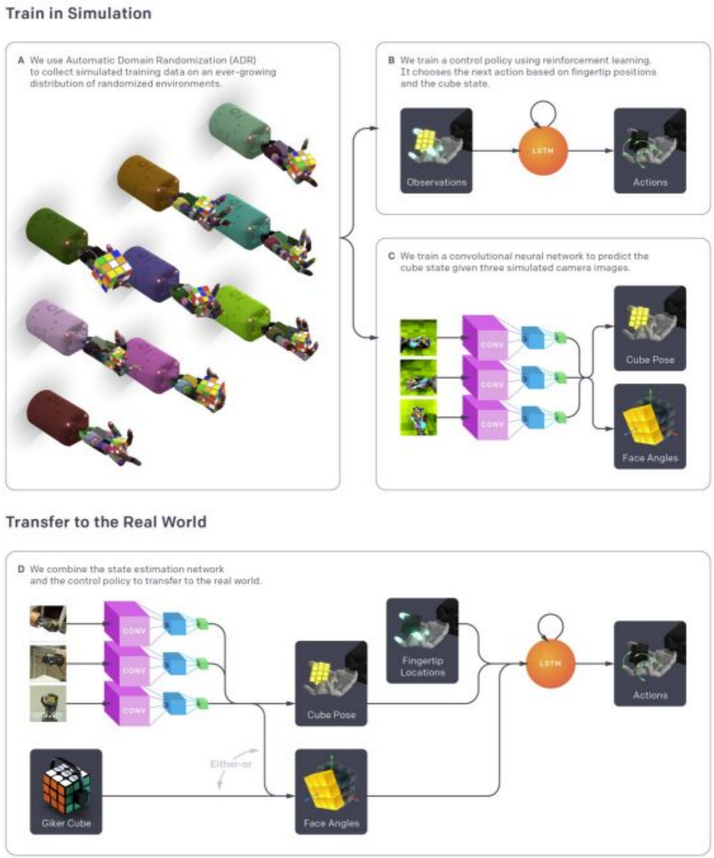


A. Domain randomization

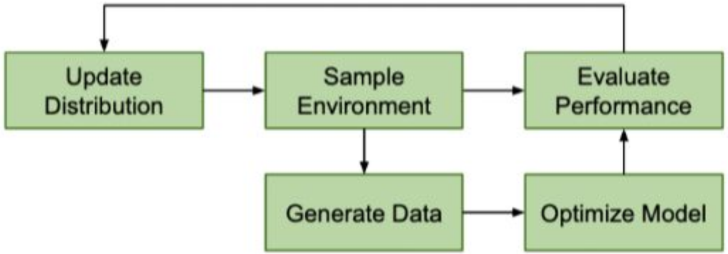
The purpose of domain randomization is to provide enough simulated variability at training time such that at test time the model is able to generalize to real-world data. We randomize the following aspects of the domain for each sample used during training:

- Number and shape of distractor objects on the table
- Position and texture of all objects on the table
- Textures of the table, floor, skybox, and robot
- Position, orientation, and field of view of the camera
- Number of lights in the scene
- Position, orientation, and specular characteristics of the lights
- Type and amount of random noise added to images

Solving Rubik's Cube with a Robot Hand

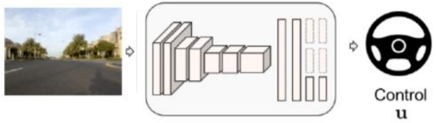


ADR: 1. gradually expand training environments (curriculum), 2. Removes need for manual domain randomization -> expansion based on performance

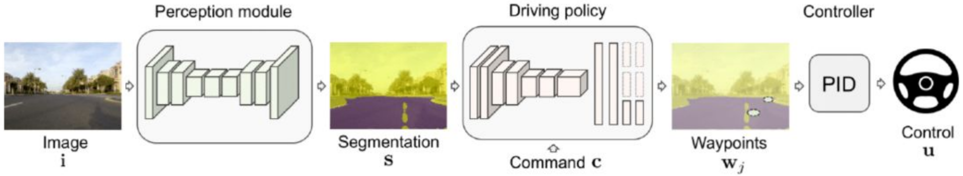


Driving Policy Transfer via Modularity and Abstraction

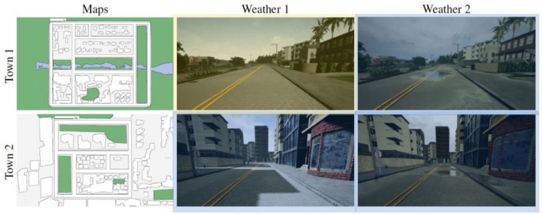
Pixels to steering wheel mapping is not SIM2REAL transferable: image textures and car dynamics mismatch



Instead: label maps to waypoint mapping is better SIM2REAL transferable: label maps and waypoints are similar across SIM and REAL. A low-level controller will take the car from waypoint to waypoint in the real world



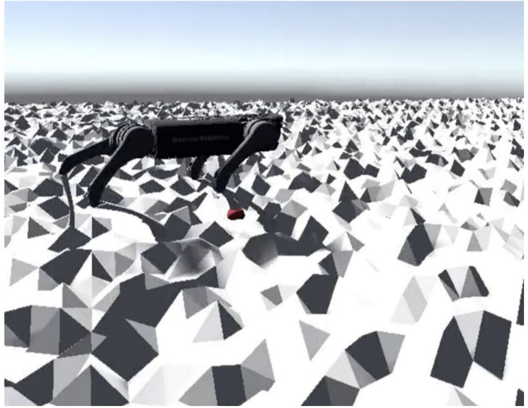
Results: Train/Test



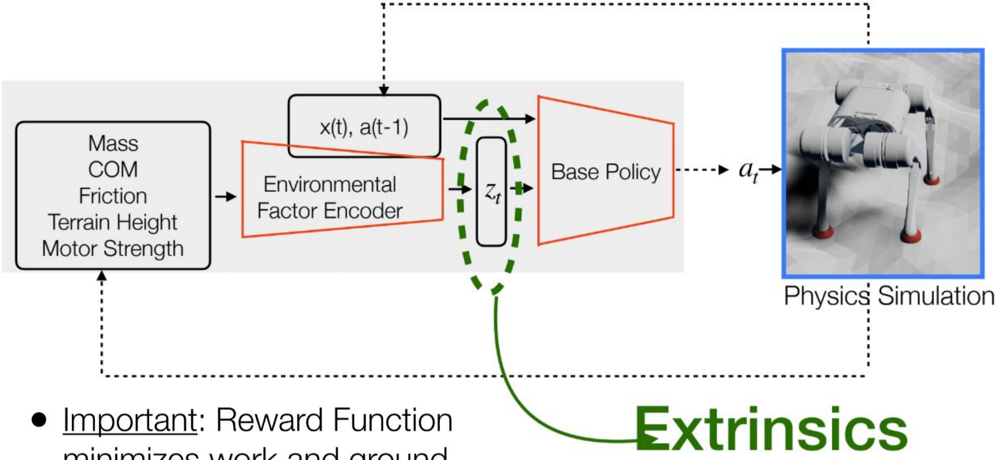
We train policies via behaviour cloning (standard regression loss) in Town1/ Weather1 dataset, and evaluate them on all four.

RMA: Rapid Motor Adaptation for Legged Robots

- Trained only on this terrain (in simulation)
- Rapidly adapts to new situations



Learn to Walk in Simulation



- Important: Reward Function minimizes work and ground impact (Biomechanics and Energetics)

RMA: Rapid Motor Adaptation for Legged Robots