IRIS: Implicit Reinforcement without Interaction at Scale

Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, Dieter Fox

Endterm Report by:

Keerthi Vasan M (ED21B034), Gokul MK (ED21B026)

ICRA 2020

Contents

1 Introduction

Learning from offline task demonstrations is a problem of great interest in robotics. For simple short-horizon manipulation tasks with little variation in task instances, offline learning from a small set of demonstrations can produce controllers that successfully solve the task. But when it comes to large scale datasets it can be problematic though large-scale supervision has accelerated progress in other fields such as Computer Vision and Natural Language Processing. New mechanisms such as RobotTurk allow for 1000s of task demonstrations to be collected in a matter of days.. The advent of such mechanisms and large datasets motivates the following question whether a policy learning algorithm necessarily need to interact or can a robust and performant policy be learned purely from external experiences in large datasets. Given this large dataset, we would like to learn a policy without allowing the policy to collect additional data. The main issues that arise in this offline training are:

- Data can consist of sub-optimal solutions and exhibit substantial diversity.
- Data can have various solution strategies. For example, a robot can pick up a soda can in different ways, such as knocking it down and grasping it from the sides, or directly grasping it from the top (Multimodal Data).

All our experiment videos are there in this Drive Link and code in this Github Link

1.1 Proposed framework

The paper proposes a new framework called **IRIS** (Implicit Reinforcement without Interaction at Scale to solve the above mentioned problems. IRIS factorizes the control problem into a goal-conditioned low-level controller that imitates short demonstration sequences and a high-level goal selection mechanism that sets goals for the low-level and selectively combines parts of suboptimal solutions leading to more successful task completions.

The paper divides the decision making process into a high-level mechanism that sets goal states for a low-level controller to try and reach. At a state s_t, the high-level mechanism selects a new goal state s_g that is held constant for the next T timesteps. Then, the low-level controller is conditioned on s_g, and is given T timesteps to try and reach that state in a closed-loop fashion. Then, control is returned to the high-level and the process repeats.

- **High Level Mechanism:** The high-level mechanism is further divided into 2 parts: a Conditional Variational Autoencoder(cVAE) that tries to model the full distribution of states $p(s_{t+T}|s_t)$ that are T timesteps away from a given state s_t. It is also used to sample a set of goal proposals. The second part is a value function $V(s)$ that is used to select the most promising goal proposal.
- **Low-Level Mechanism:** The low level controller is a Recurrent Neural Network(RNN) that outputs an action a_t at each timestep, given a current observation s_t and goal s_g.

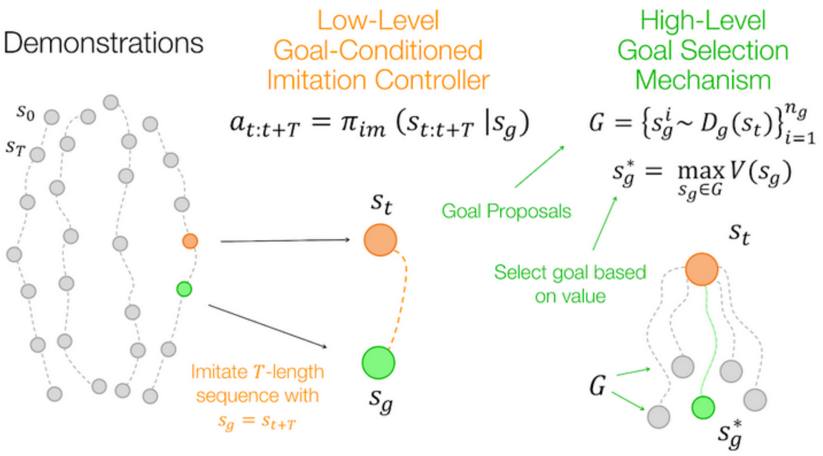

Demonstrations
s_0
s_T
Low-Level Goal-Conditioned Imitation Controller
$a_{t:t+T} = \pi_{im}(s_{t:t+T} | s_g)$
s_t
s_g
Imitate T-length sequence with $s_g = s_{t+T}$
High-Level Goal Selection Mechanism
$G = \{s_g^i \sim D_g(s_t)\}_{i=1}^{n_g}$
$s_g^* = \max_{s_g \in G} V(s_g)$
Goal Proposals
Select goal based on value
s_t
G
s_g^*

Figure 1: IRIS

1.1.1 Low-Level Controller

The low-level controller is a goal-conditioned RNN $\pi_\theta(s \mid g)$, trained on trajectory sequences of length T. Consecutive state-action sequences $(s_t, a_t, \ldots, s_{t+T-1}, a_{t+T-1}, s_{t+T})$ are sampled from trajectories in the dataset. The last observation in each sequence, s_{t+T}, is treated as a goal that the RNN should try to reach, and the RNN is trained to output the action sequence $a_t, a_{t+1}, \ldots, a_{t+T-1}$ from the state sequence $s_t, s_{t+1}, \ldots, s_{t+T-1}$ and the goal $s_g = s_{t+T}$.

The loss function for the RNN is a simple **Behavioral Cloning** loss:

$$\boxed{L_\theta(a_{t:t+T}, s_{t:t+T}) = \sum_{k=t}^{t+T-1} ||a_k - \pi_\theta(s_k \mid s_g)||_2^2}$$

1.1.2 High-Level Goal Selection Mechanism

The high-level goal selection mechanism chooses goal states for the low-level controller to reach. It consists of two components: (1) a conditional Variational Autoencoder (cVAE) $(E_\phi(s_g, s), D_\phi(z, s))$ to propose goal states at a particular state, and (2) a value function $V(s_g)$ that models the expected return of goal states.

The cVAE is a conditional generative model trained from trajectories in the dataset. An encoder maps a current and future observation to the parameters of a latent Gaussian distribution, $\mu_g, \sigma_g = E_\phi(s_{t+T}, s_t)$, and the decoder is trained to reconstruct the future observation from the current observation and a latent variable sampled from the encoder distribution:

$$\hat{s}_{t+T} = D_\phi(z, s_t), \quad z \sim \mathcal{N}(\mu_g, \sigma_g)$$

The encoder distribution is regularized with a KL-loss:

$$KL(\mathcal{N}(\mu_g, \sigma_g) \mid\mid \mathcal{N}(0, 1))$$

with weight β_g to encourage the encoder distribution to match the prior latent distribution $p(z) = \mathcal{N}(0,1)$. At test time, the decoder acts as a conditional generative model by sampling latents $z \sim \mathcal{N}(0,1)$ and passing them through the decoder.

The value function is a state-action value function $Q_\psi(s,a)$, trained using a variant of Batch Constrained Q-Learning (BCQ). The loss function for the value network is a modified BCQ update:

$$\boxed{L_\psi(s,a,r,s') = (Q_\psi(s,a) - Q_{\text{target}})^2}$$

where the target value is computed by considering a set of action proposals from the cVAE $A = \{D_\omega(z,s) \mid z \sim \mathcal{N}(0,1)\}_{i=1}^{M}$ and maximizing the Q-network over this set of actions:

$$Q_{\text{target}} = r + \gamma \max_{a_i \in A} Q'_\psi(s', a_i)$$

Algorithm 1 IRIS: Train Loop

```
Require: $\pi_\theta(s \mid s_g), \{E_\phi(s_g,s), D_\phi(z,s)\}, Q_\psi(s,a), \{E_\omega(a,s), D_\omega(z,s)\}$   ▷ Policy, Goal cVAE, Value Network, Action cVAE
 1: for $i = 1, 2, \dots, \text{iter}$ do
 2:     $(s_t, a_t, r_t, s_{t+1}, \dots, s_{t+T-1}, a_{t+T-1}, r_{t+T-1}, s_{t+T}) \sim \mathcal{D}$   ▷ Sample $T$-length sequence from the dataset
 3:     $s_g \leftarrow s_{t+T}$   ▷ Treat last observation as goal
 4:     $\hat{a}_t, \hat{a}_{t+1}, \dots, \hat{a}_{t+T-1} \leftarrow \pi_\theta(s_t, \dots, s_{t+T-1} \mid s_g)$   ▷ Goal-conditioned action sequence prediction from RNN Policy
 5:     $\theta \leftarrow \arg\min_\theta \sum_t^{t+T-1} \|\hat{a}_t - a_t\|_2^2$   ▷ Update policy with imitation loss
 6:     $\mu_g, \sigma_g \leftarrow E_\phi(s_g, s_t)$, $z \sim \mathcal{N}(\mu_g, \sigma_g)$
 7:     $\phi \leftarrow \arg\min_\phi \|s_g - D_\phi(z, s_t)\|_2^2 + \beta_g KL(\mathcal{N}(\mu_g, \sigma_g) \parallel \mathcal{N}(0,1))$   ▷ Train Goal cVAE to predict goals
 8:     $\mu_a, \sigma_a \leftarrow E_\omega(a_{t+T-1}, s_{t+T-1})$, $z \sim \mathcal{N}(\mu_a, \sigma_a)$
 9:     $\omega \leftarrow \arg\min_\omega \|a_{t+T-1} - D_\omega(z, s_{t+T-1})\|_2^2 + \beta_a KL(\mathcal{N}(\mu_a, \sigma_a) \parallel \mathcal{N}(0,1))$   ▷ Train Action cVAE on last action
10:     $A \leftarrow \{a_i \sim D_\omega(z, s_{t+1})\}$   ▷ Set action proposals from Action cVAE
11:     $\hat{Q} \leftarrow r_{t+T-1} + \gamma \max_{a_i \in A} Q'_\psi(s_{t+1}, a_i)$   ▷ Set target value for value update
12:     $\psi \leftarrow \arg\min_\psi (\hat{Q} - Q_\psi(s_t, a_t))^2$   ▷ Update value network
13: end for
```

1.2 Implementation

We split the whole dataset into a list of trajectories. Instead of subsampling, as mentioned in the paper, we used this entire trajectory. We use LSTM instead of an RNN controller. Till our midterm report, we were using just two modules: a low-level policy controller and goal proposal VAE. The paper proposes to use the value function to select goals from the list of goals provided by the goal proposal VAE.In our final report, we include this module and conduct a study on how well the proposed algorithm works in the presence of the value function.

The paper conducted studies in a simple graph reach environment, Robosuite Lift Environment and finally on the Roboturk Can Datasets. The experiments presented in the paper were accuracy-based performance comparisons since they were done using an offline learning algorithm. Our goal from start was to first code the algorithm since it wasn't available to us and check in a simple environment and execute this algorithm in the Graph Reach and Robosuite Lift dataset. The Robosuite Turk was a very big dataset which our laptops couldn't handle.

To mimic a similar graph reach environment(which is not made available by the authors), we used a simple point maze environment with no obstacles. Our majority of the time was spent in tuning our code to work on the lift dataset. Since the environment is not as easy as the point maze environment and also since the codes were not provided in the paper, more tuning was required from the architecture of the networks used. In our analysis, we were able to achieve 80-90% success in the point maze, which was similar to the performance the authors obtained in theirs, which was also close to this(but they did it for 100 trials).

2 Experiments(Midterm)

2.1 Experiment 1: Pointmaze U maze

This experiment was conducted on the D4RL dataset using the Pointmaze environment with a U-shaped maze configuration. The reward structure was sparse, providing a reward of 1 only when the agent successfully reached the goal and 0 otherwise. The agent was initialized at random start positions and the task required the green ball (agent) to navigate and reach the red ball (goal) from these randomized states. The latent dimension used for the autoencoder was 8. The lstm had one state vector. The low level policy controller was given 3 environment steps to reach the goal before transferring control back to the goal cVAE. In this environment we were able to achieve 100% success rate on 10 trials.

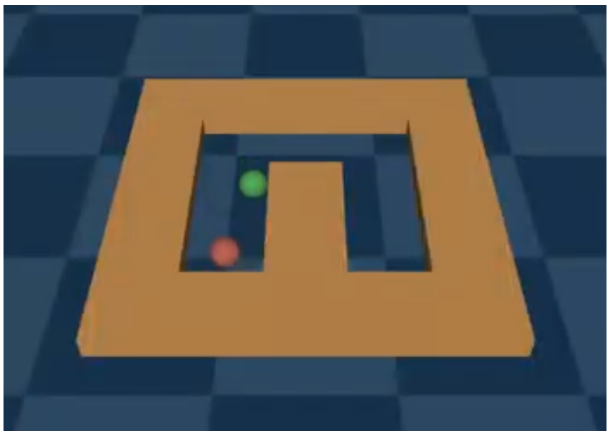


Figure 2: Pointmaze U-maze

3 Experiments(Endterm)

3.1 Experiment 1: Pointmaze maze2d Open

The experiment utilized the D4RL dataset in the Pointmaze environment with a maze2d-open configuration. The reward structure was sparse with a reward of 1 only when the green ball (agent) successfully reached the red ball (goal) from a random start state. The agent was initialized at random start positions. With an action step size of 2 and value function-based sub-goal iterations set to 5, the approach achieved an accuracy of 80% over 20 trials.

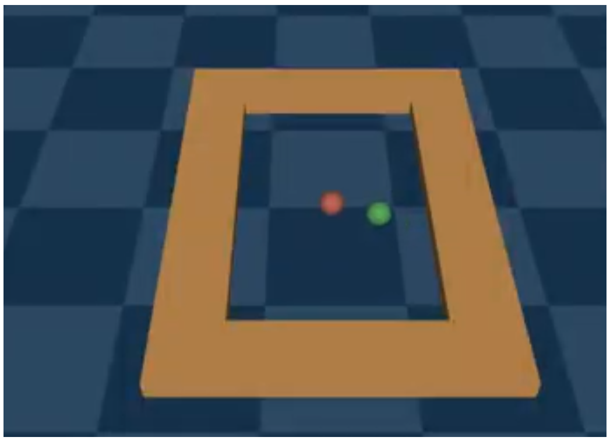
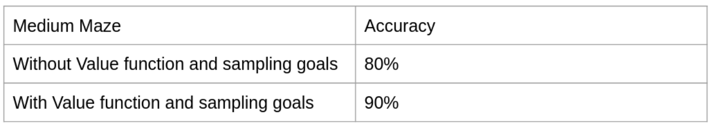


Figure 3: Pointmaze maze2d Open

3.2 Experiment 2: Maze2d medium v1

This experiment was conducted using the D4RL dataset in the Maze2d-medium-v1 environment. The reward was sparse, providing a value of 1 only when the green ball (agent) reached the red ball (goal) from a randomly initialized start state. The low level policy had an action step size of 2 and value function-based sub-goal iterations set to 5, the approach achieved an accuracy of 90% across 20 trials.

We also experimented with the use of value function in selecting the best goal proposal in this environment. One approach was to use the naive Conditional VAE to choose a single goal and let the LSTM decide the trajectory. In yet another approach was to use Value function to select the best goal from a set of proposals from conditional VAE based on expected reward. As expected, the accuracy in the latter approach was higher than the former one.



Medium Maze	Accuracy
Without Value function and sampling goals	80%
With Value function and sampling goals	90%

Figure 4: Value function experiments

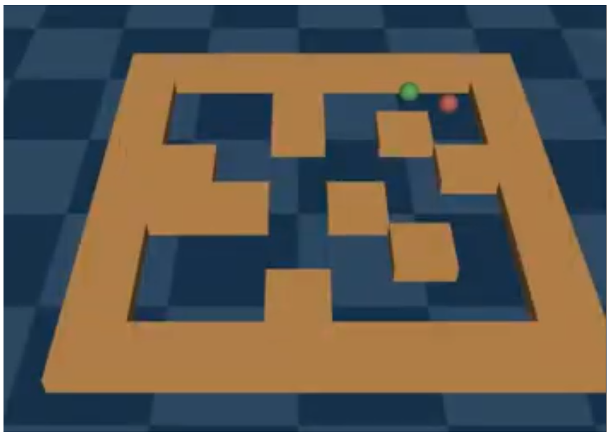


Figure 5: Maze2d medium v1

3.3 Experiment 3: Robosuite Lift

This experiment was conducted using the RoboMimic low-dim-lift dataset in the Robosuite Lift task environment. The reward was sparse, given only when the robotic arm successfully lifted the object above a specified height. While the accuracy was suboptimal, the robotic arm exhibited expected behavior, demonstrating its ability to lift the object. Improved accuracy could likely be achieved with better network tuning(the architecture was not provided in the paper) and optimization.
We feel that this can be because of 2 reasons:

- The value functions was not trained properly , a better network tuning should have been performed
- The dataset might have contained good trajectories which might have helped thee goal Cvae itself to provide good goals and the not properly trained value function might have been hindering it.

We cant conclude that the value function is not require since we could observe that the agent was able to perform better in the previous environments.

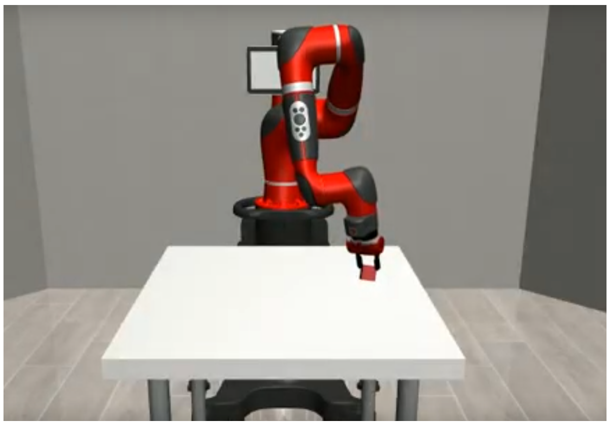


Figure 6: Robosuite Lift

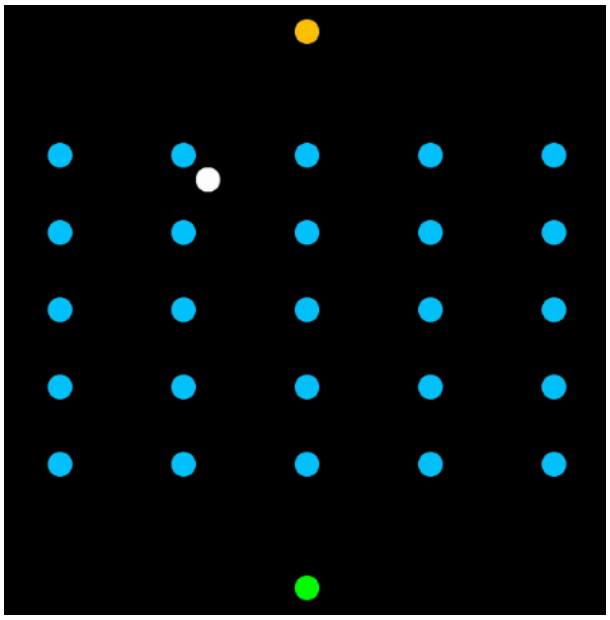


Figure 7: Graph Reach

3.4 Degree of Suboptimality

The paper states that the proposed algorithm will achieve near-optimal performance from offline trajectories that contain multimodality. We wanted to verify this by collecting data from a custom graph reach environment by varying the degree of suboptimality, as no other lite dataset available on the web would suit this. We created this environment but ran into issues because we initially thought this was a discrete state space, and action space was also discrete, which the agent had to select from the nearest dots. But while training the policies, we got exploding gradients and could not train. We figured out this was basically because of the discrete assumption. The time wasn't enough to do this task(we proposed do in the midterm) since the entire time was spent on tuning on robosuite lift task, but we figured out to configure the environment like this:

- The agent is to have an action space from 0 to 2π
- The state space is basically a filled square
- When an action is selected, the agent moves by delta u, which is a hyperparameter
- For introducing a degree of suboptimality, we set a variable p(probability), like the epsilon greedy method, we choose the optimal path with probability (we choose to go straight), with 1-p, we choose a suboptimal(We do a random a walk, and after some time, we execute the optimal path from the final state of the random walk.

4 Further Ideation

In real-world robotic tasks, it is easier to record a dataset of image observations, rather than to record the state trajectories and motor values from the tele-operated robot. Recent works on learning world models from pixels have been rapidly explored, the first being the PlaNet from Google. Another pioneer research was DREAMER, which was built on top of PlaNet. We planned to integrate the "learning latent dynamics" and "latent imagination and planning" from DREAMER into the IRIS framework to tackle the aforementioned Offline dataset drawbacks.

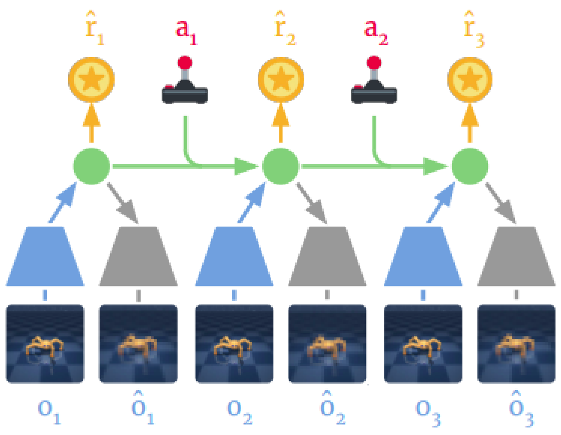


Figure 8: Dreamer Architecture

Dreamer is an online algorithm that learns latent dynamics from pixel inputs, collecting experiences while simultaneously training both a world model and an action model.Dreamer was chosen because,

- Using image pixels as input data is simpler and more accessible than recording state trajectories, which can be tedious.
- Its imagination-based approach refines action and value models, making it well-suited for robotics and control tasks.
- Additionally, Dreamer's latent dynamics and recurrent architecture enable it to handle high-dimensional, complex tasks effectively.

4.1 Architecture Details

The proposed architecture aims to run IRIS in the latent space after the world models have been learned.

- The planned setup involves using an offline dataset of episodic image observations and actions.
- Dreamer's recurrent RSSM (Recurrent State-Space Model) policy remains consistent, operating within a latent state space. In this setting, each sequence's final observation serves as a goal for training the dynamics.
- A high-level goal-selection mechanism then chooses a target in the latent space for the low-level RNN policy to reach. The system utilizes a conditional variational autoencoder (cVAE) to propose potential goals.
- For each goal, the RNN generates an imagined plan, and the value model evaluates these plans by calculating the expected return, ultimately selecting the optimal goal path.

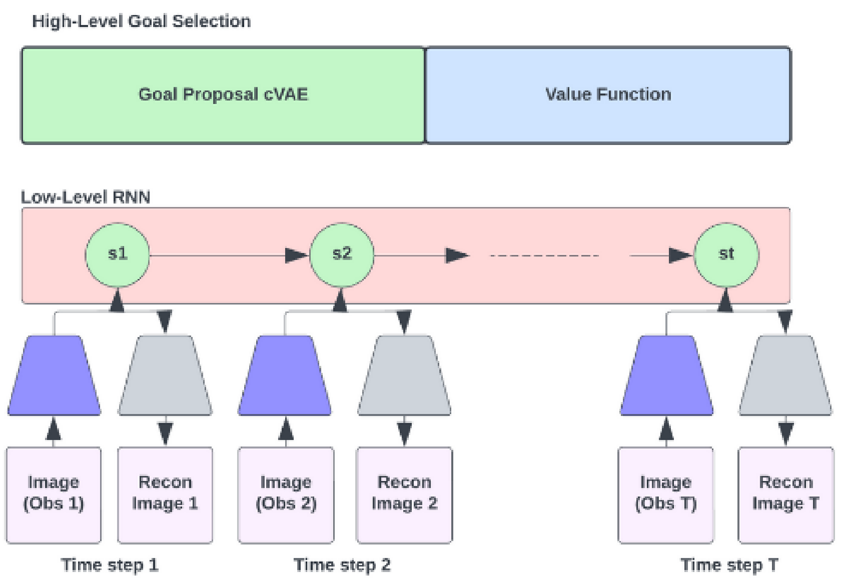


Figure 9: Proposed Architecture

4.2 Training Details

The training is done for each component with their respective loss function.

- The low-level RNN is trained using a composite loss function that includes behavior cloning, reconstruction loss, and a KL regularizer to align the transition model with the representation model. This multi-faceted training approach ensures that the RNN captures both the dynamics and structure of the environment.
- The goal proposal mechanism is implemented using a conditional Variational Autoencoder (cVAE), a generative model trained on pairs of current and future observations. These observations are sampled from the encoded latent states generated as the RNN follows its trajectory.
- The value function complements this system by estimating returns for all states along imagined trajectories. It regresses these value estimates, providing a mechanism to evaluate and select the most promising plans generated during imagination-based exploration.

References

[1] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg and Dieter Fox, *IRIS: Implicit Reinforcement without Interaction at Scale for Learning Control from Offline Robot Manipulation Data*, 2020.

[2] Danijar Hafner, Timothy Lillicrap, Jimmy Ba and Mohammad Norouzi, *Dream to Control: Learning Behaviors by Latent Imagination*, 2020.

[3] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee and James Davidson, *Learning Latent Dynamics for Planning from Pixels*, 2019.