LOCALLY LINEARIZED NEURAL NETWORK DYNAMICS AND COST LEARNING IN LATENT SPACE FOR TRAJECTORY OPTIMIZATION IN MODEL-BASED REINFORCEMENT LEARNING

Kyle Stachowicz, Kai Yun, Haoming Guo
University of Califonia, Berkeley
{kstachowicz, kaiyun799, mike0221}@berkeley.edu

ABSTRACT

Traditional trajectory optimization algorithms fall into two classes: gradient-based algorithms and gradient-free algorithms. While gradient-based methods are relatively efficient, especially with long-horizon planning problems, they tend to struggle when the optimization cost landscape is non-smooth, which often occurs with dynamics learned using neural networks. Prior work in the embed-to-control (E2C) framework has introduced a framework for learning latent dynamics and approximate Jacobians using a variational autoencoder [1].

In this work, we construct a loss function that enforces explicit local linearity conditions so that the learned Jacobian dynamics can produce better results when used with gradient-based trajectory optimization methods, and give a reparametrization of the Jacobian dynamics to allow higher-order linearized dynamical phenomena than the rank-1 dynamics proposed by the original E2C paper. We also discover that when paired with a low-dimensional latent space, the VAE-based formulation originally proposed in E2C is susceptible to local singularities and typically results in an entangled latent space. We borrow concepts from topology to show that the latent dimension should be larger than the true state dimension of the system, and provide a condition for when the optimizer should be able to successfully disentangle the latent space into a single continuous embedding with no singularities or discontinuities. Additionally, we develop a method for learning locally-quadratic costs by making use of locally-linear residual functions whose linearizations can be used to construct a Gauss-Newton approximation of the second-order expansion of the costs for use in gradient-based optimizers. Finally, we implement our proposed changes and test them against a toy system. We show that our changes allow an iLQR controller to find a valid swing-up trajectory for a pendulum, while the same controller operating on the latent structure learned by the unmodified algorithm is unable to do so due to poor latent space structure.

o_t	Observation at time t
$\tilde{o}_t$	Reconstructed observation o_t
u_t	Control input at time t
z_t	Latent state at time t
$\bar{z}$	Linearization point
μ_{enc}	Encoder neural network for mean
σ_{enc}	Encoder neural network for standard deviation
$Q(z_t \vert o_t)$	Probability distribution $\mathcal{N}(\mu_{enc}(o_t), \sigma_{enc}(o_t))$
$Q'(z_{t+1} \vert o_t, u_t)$	Probability distribution $\mathcal{N}(A\mu_{enc}(o_t) + Bu_t + f_0, A\sigma_{enc}(o_t)A^T)$
ϕ_{dec}	Decoder neural network

Table 1: Overview of notations used throughout this paper.

I Introduction

Model-based reinforcement learning algorithms use trajectory optimizers with learned dynamics models to enable efficient and effective control of complex systems. There are two main types of trajectory optimizers: sampling-based and gradient-based. Gradient-based methods are efficient but often fail with non-smooth dynamics, such as those found in neural networks (i.e. ReLU activation functions). A common gradient-based method is the iterative linear quadratic regulator (iLQR), which repeatedly approximates the system near a reference trajectory using linear dynamics and then solves for the optimal controller.

Sampling-based methods, on the other hand, work well with neural networks but are computationally expensive. One such method is the model predictive path integral (MPPI), which rolls out many samples forward and then picks an exponentially-weighted average. These methods also tend to exhibit poor performance on long-horizon tasks due to the curse of dimensionality.

Latent variable models are often used in reinforcement learning, as the true state of an underlying system is not always known or specified a priori by the system designer. For example, in learning to control from pixels, the learning process must jointly discover a latent representation of the state, an encoder to encode latent states from observations, and a dynamics model operating in the latent space[1].

One popular latent variable model is the variational autoencoder (VAE), which is a generative model that uses a neural network to approximate the distribution of data [2]. The VAE encodes observations into a latent space and then decodes them back into the original space, allowing for efficient representation and manipulation of high-dimensional data.

Prior work (Embed to Control or E2C) has proposed learning latent space dynamics and approximate jacobians, designed to work well with iLQR. In particular, this method attempts to learn low-dimensional latent representations, which is beneficial for LQR [1]. However, it is only usable as a goal-reaching method, as it does not provide a method of learning the quadratic costs required by LQR.

II Contribution

In this paper, we present an analysis of the failure modes of the embed to control (E2C) method for model-based reinforcement learning. We show that the learning process used by E2C is not guaranteed to yield a continuous latent space, which can cause problems for trajectory optimizers like iterative linear quadratic regulator (iLQR).

Additionally, we demonstrate that the learned jacobian used by E2C is trivial and has rank-1 in all dimensions except 1. This means that the learned jacobian is not guaranteed to match the true jacobian of the learned dynamics, which can lead to suboptimal control.

To address these issues, we present a method for improving jacobian estimates from E2C with explicit error terms. By adding these error terms, we can better approximate the true jacobian of the learned dynamics, resulting in more accurate control.

Furthermore, we propose a method for learning second-order expansions for costs from residuals. This allows us to incorporate additional information about the underlying system into our optimization process, leading to better control performance.

Overall, our contributions provide a deeper understanding of the limitations of E2C and offer solutions for overcoming these limitations to improve model-based reinforcement learning with gradient-based trajectory optimizers.

III Related Work

A. Model-based Reinforcement Learning

Reinforcement learning can be categorized as either *model-free* or *model-based* based on how the agent makes decisions. While the agent learns the control policy directly in model-free RL, the agent utilizes a model of the dynamical system of the environment to make decisions. Model-based algorithms are sample-efficient in theory, but it struggles in modeling with complex function approximators such as neural networks. For model-based algorithms to perform well, i.e., model the system well, it needs to mitigate the distributional shift problem by collecting new data, but that means the model-approximator needs to be good early on in training even when it does not have enough data. Neural networks perform well in the high data region, but struggle otherwise. Thus, model-based algorithms tend to perform poorly in the early stages and thus do not produce effective exploration which leaves them stuck. Utilizing uncertainty-aware models such as Bayesian neural networks can help since they may avoid exploiting the model. Bootstrap ensemble is

another method that helps mitigate this problem by training multiple independent models with independent training sets but from the sample distribution. PETS (Probabilistic Ensembles with Trajectory Sampling) is a method that combines uncertainty-aware deep network dynamics models with bootstrapped ensembles to reach performance comparable to the asymptotic performance of state-of-the-art model-free and model-based algorithms [3].

Back-propagating through the learned model into the policy to optimize the controller often results in parameter-sensitivity problems and vanishing/exploding gradients problems. A solution to such issues is to utilize model-free RL algorithms applied to the learned model, using it to generate additional training data. Further improvement to this solution includes making short rollouts with the learned model to avoid the exponential increase in modeling error that accompanies long-horizon rollouts [4].

Model-based reinforcement learning can be applied to a system with complex observations by incorporating latent-space models:

$$\max_{\phi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_t \sim Q, s_{t+1} \sim Q'} [\log p_\phi(z_{t+1,i}|z_{t,i}, u_{t,i}) + \log p_\phi(o_{t,i}|z_{t,i})] \tag{1}$$

where $o_{t,i}$, $z_{t,i}$, and $u_{t,i}$ are the observation of the state, latent state representation, and action at timestep t for a trajectory i. This formulation is adjusted and concretized in the embed-to-control framework, discussed in B.

In this paper, we use off-policy, offline data to learn a system's latent dynamics given rendered images of an environment. Here, the rendered images are the partially observed state of the system. The environment goes through random steps and the images are stacked to satisfy the Markov property while allowing the model to infer position, velocity, and acceleration. While collecting data, most of the state-space is covered by ensuring that the environments are uniformly reset over the entirety of the plausible state-space once the *done* condition is met. Here, 'plausible state-space' is defined as a safe and physically possible initial state for a given environment. The Jacobian dynamics and cost are learned from the latent space of this training set with the methodologies outlined in Section IV.

B. Embed-to-control (E2C)

The Embed to Control (E2C) framework is a variational framework for jointly learning an autoencoder-decoder that translates between pixel observations and a latent space as well as a transition function to compute low-dimensional dynamics in that learned latent space. The encoder-decoder architecture is based on a variational autoencoder, which regularizes the latent space to be approximately Gaussian. The encoder function outputs not only a mean but also a variance, together parameterizing a Gaussian in latent space. Latent space vectors sampled from this distribution can be mapped back to observations via the decoder:

$$z_t \sim \mathcal{N}(\mu_{enc}(o_t), \sigma_{enc}(o_t)) \tag{2}$$

$$\tilde{o}_t = \phi_{dec}(z_t) \tag{3}$$

The transition function is explicitly written as a linearized dynamical system around some point $\bar{z}$ in latent space. Let z_t be the previous state and u_t be the control; the transition is then written as follows:

$$z_{t+1} = A(\bar{z})z_t + B(\bar{z})u_t + f_0(\bar{z}) \tag{4}$$

Note that A and B are only written as functions of $\bar{z}$, the linearization point. While the linearization could also be taken around a fixed action u, this is typically unnecessary in trajectory optimization for physical system because real-world systems are often *control-affine* (the response is a linear function of the control, given the state). When sampling forwards through the dynamics during training, $\bar{z}$ is taken to be equal to z_t, which creates additional terms in the true Jacobian. This is addressed in detail in subsection C..

The E2C loss function contains the standard VAE ELBO terms for regularization and reconstruction, as well as an additional consistency loss to ensure multi-step rollouts can be computed in latent space without encoding and decoding every step:

$$\begin{aligned} \mathcal{L}_{e2c}(o_t, u_t, o_{t+1}) = \mathop{\mathbb{E}}_{\substack{z_t \sim Q(z_t|o_t), \\ z_{t+1} \sim Q'(z_{t+1}|o_t, u_t)}} \Big[& \mathcal{L}_{recon}(\tilde{o}_t, o_t) + \mathcal{L}_{recon}(\tilde{o}_{t+1}, o_{t+1}) + \\ & KL[Q'(z_{t+1}|o_t, u_t)||Q(z_{t+1}|o_{t+1})] \Big] + KL[Q(z_t|o_t)||\mathcal{N}(0, I)] \end{aligned} \tag{5}$$

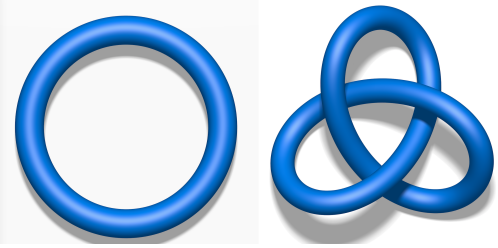

Figure 1: Unknot and trefoil knot. These are both the same manifold embedded in a 3D latent space, but one cannot be turned into the other via continuous deformation. With an extra 4th dimension, there is no such restriction.

C. Iterative Linear Quadratic Regulator (iLQR)

iLQR is used for local optimal feedback control of nonlinear systems [5]. The method iteratively uses LQR to find the optimal trajectory. In essence, it uses Newton's method for optimization to refine the trajectory until it converges to the optimal solution. Given a system dynamics $x_{t+1} = f(x_t, u_t)$ and cost function $c(x_t, u_t)$, iLQR is an approximation of Newton's method for solving the following [6]:

$$\min_{u_1,\ldots,u_T} c(x_1, u_1) + c(f(x_1, u_1)) + \cdots + c(f(f(\cdots)\cdots), u_T). \tag{6}$$

Newton's method uses both the first-order and second-order derivative to create a quadratic approximation of the system dynamics, f. In this paper, we use the learned Jacobian dynamics as the first-order approximate derivative of the system dynamics. This is discussed in more detail in Section IV, subsection C. Additionally, a cost (negative reward) function $\ell(x, u)$ is learned in the latent space. This function can be used as a Gauss-Newton approximation for the Hessian, which can be used as the second-order approximate derivative of the system dynamics. This is discussed in more detail in Section IV, subsection D.

IV Improvements to E2C

A. Latent Dimension d

In the experiments presented in the original E2C paper, the latent dimension is typically set to be the latent dimension of the underlying task. For example, in the planar motion problem, the latent dimension is set to 2, as the system's state can be written as just an (x, y) pair. In the pendulum problem, the latent dimension is set to 3 – despite the minimal coordinates for the system being $(\theta, \dot{\theta})$ – because the θ variable "loops" back on itself; in mathematical terms the minimum dimension into which it can be *continuously embedded* is 3 (i.e. $(\cos\theta, \sin\theta, \dot{\theta})$).

However, this does not imply that the learning process will succeed in these minimal coordinates. Indeed, we find that choosing minimal coordinates is actively harmful. This can be explained through topology, the mathematical field concerned with continuity and invariance under continuous deformation. Assume we have a latent space that is shaped like some subset of an n-dimensional sphere. For example, both the $(\theta, \dot{\theta})$ and (x, y) manifolds are subsets of S^2, the sphere surface (note that we often think of this as a 3-dimensional sphere, but it is hollow and for topological purposes it is a 2-dimensional manifold).

First, let us refer to the analogy of a 1-dimensional sphere (this is just a circle, or a loop). In three dimensions, a loop can be tied in knots that are materially different from the original shape, such that they *cannot be untied* without cutting the loop. Figure 1 shows an unknotted loop and a trefoil knot, one such example. However, when embedded in 4 dimensions, the same knot can be trivially undone without cutting or crossing edges: one edge can be moved "through" another, as viewed in three dimensions, by first moving it a small amount in a fourth dimension, moving it past the crossing point, and bringing it back into the original space.

Now, let us return to our original problem. The optimizer is only able to make small steps (equivalent to the constraint of no cutting), and at all points no two real states should map to the same latent vector (no crossings). A "knotted" representation has additional regions of high curvature. Worse yet, in practice the embedding is not necessarily one-to-one, and so there may be singularities at which the embedding is degenerate.

For manifolds of higher dimension $n > 1$ embedded in a latent space of dimension d, it can be guaranteed that the sphere S^n (and its subsets) has no knots when $d \geq 2n + 1$.

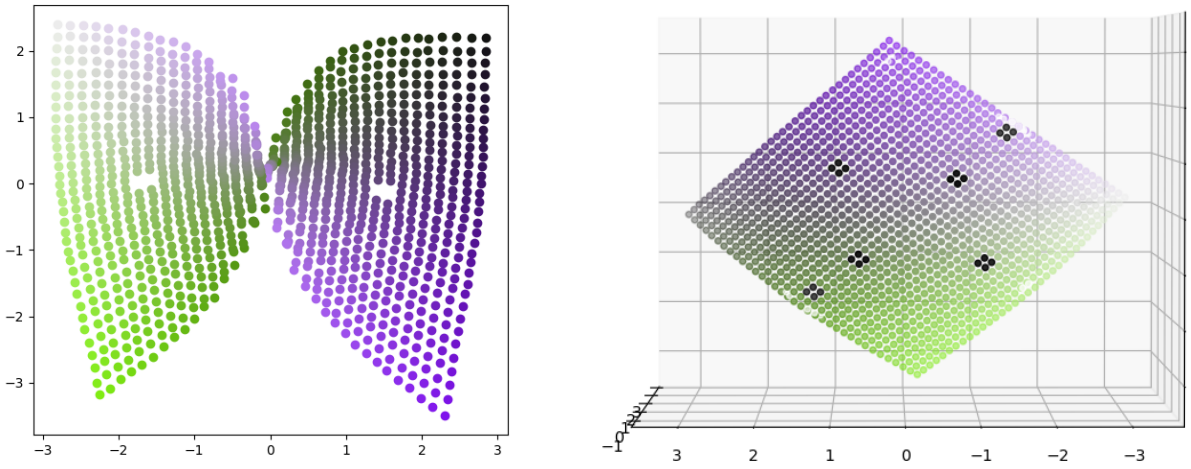

Figure 2: Left: embedding of planar dynamics in 2D latent space, with a singularity introduced by random initialization. This latent space cannot be continuously deformed into a plane in two dimensions without crossing itself (which would increase reconstructive loss) and so this forms a local minimum. Right: embedding in 3D latent space, without a singularity.

To empirically validate these claims, we trained a standard implementation of E2C on a planar motion task with intrinsic dimension $n = 2$. Using latent dimension 2 often results in singularities (see Figure 2).

B. Jacobian Rank

The original E2C paper parametrizes the system dynamics matrix a rank-1 update to the identity matrix:

$$A(\bar{z}) = I + v(\bar{z})r(\bar{z})^T \tag{7}$$

This is intended to allow the network to predict fewer parameters. However, it severely limits the expressive capacity of the resultant Jacobian: a linear system is characterized by the eigenvalues of A, and in parametrization only one eigenvalue differs from identity. This precludes several fundamental types of dynamics, including oscillators, integrator chains, and saddle-point modes.

We consider two approaches to avoid this issue. In the first, A is predicted as a rank-k (rather than a rank-1) update, where k is some design parameter chosen to be smaller than the latent dimension:

$$A(\bar{z}) = I + \sum_{i=1}^{k} v_i(\bar{z})r_i(\bar{z})^T \tag{8}$$

The second parametrization predicts a block-diagonal form of A. Each 2×2 block can either have two distinct real eigenvalues or a complex conjugate pair, meaning that $eig(A)$ can produce all possible combinations of eigenvalues. In conjunction with a linear transformation at the encoder and decoder, this parametrization is complete (parametrizes all possible dynamics matrices).

C. Jacobian Approximations

Consider the dynamics during the training process, where $\bar{z} = z_t$. The true Jacobian is:

$$\frac{\partial z_{t+1}}{\partial z_t} = \underbrace{A(\bar{z})}_{\text{E2C approx.}} + \underbrace{\frac{\partial A}{\partial \bar{z}} \cdot z_t + \frac{\partial B}{\partial \bar{z}} \cdot u_t + \frac{\partial f_0}{\partial \bar{z}}}_{\text{neglected terms}} \tag{9}$$

Without these terms the dynamics fail to compute the true Jacobian. In the pathological case, all of the dynamics could be computed in $f_0(\bar{z})$, leaving $A = 0$ and implying static dynamics.

This shortcoming can be addressed by evaluating the dynamics both at $\bar{z}$ and z_t and constraining the outputs to be similar via an additional loss term:

$$\mathcal{L}_{jac} = \mathbb{E}\left[A(\mu)z_t + B(\mu)u_t + f_0(\mu) - A(z_t)z_t - B(z_t)u_t - f_0(z_t)\right]^2 \tag{10}$$

D. Cost Learning

Many model-based RL algorithms model dynamics in some known state space and use a known reward function in this state space. However, with latent dynamics no such ground-truth state space exists and thus rewards must be learned together with the latent space. To avoid learning rewards (which, in the iLQR setting, requires learning a quadratic approximation), E2C writes its rewards in latent space (note the change in sign; this converts the problem to learning costs rather than rewards):

$$-r(z,u) = \ell(z,u) = (z - z_{target})^T Q(z - z_{target}) + u^T R u \tag{11}$$

We extend E2C by considering *residual models* of the costs. We write the cost function as the square of some locally-linear vector function:

$$\ell(z,u) = \|G(\bar{z})z + H(\bar{z})u + r_0(\bar{z})\|^2 \tag{12}$$

Assuming local linearity and taking the Gauss-Newton approximation (which ensures positive-definiteness, acts as a regularizer, and simplifies calculations), this yields the following second-order Taylor expansion around $\bar{z}$:

$$\ell(z,u) \approx \frac{1}{2}\begin{bmatrix}(z-\bar{z})^T & u^T & 1\end{bmatrix}\begin{bmatrix}G^TG & G^TH & G^Tr_0 \\ H^TG & H^TH & H^Tr_0 \\ r_0^TG & r_0^TH & r_0^Tr_0\end{bmatrix}\begin{bmatrix}z-\bar{z} \\ u \\ 1\end{bmatrix} \tag{13}$$

This Gauss-Newton approximation can be used directly in the iLQR algorithm.

V Experimental Results

In addition to the planar embeddings generated in Figure 2, we also trained our formulation of the model on an inverted pendulum task from pixels. This task has unstable dynamics and more complicated latent structure: because angles wrap from 0 to 2π, the latent space should form a closed band. Because this is still a 2-manifold, we expect that it should unknot in $2 \cdot 2 + 1 = 5$ latent dimensions. Indeed, Figure 4 shows three projections of the latent space onto two-dimensional slices, and we see that the latent space exhibits the desired structure.

Figure 3 shows the component loss curves for our proposed modifications to E2C on the pendulum task with pixel observations. The lower-dimensional latent space ($d = 4$) has relatively good reconstruction but poor dynamics consistency: this likely stems from the fact that the embedding of the state manifold in $\mathbb{R}^4$ ends up being discontinuous and self-intersecting, meaning that most points will have low training error while states near these problematic regions will have very high consistency loss. Amongst the runs with higher-dimensional latent space, it is evident that increasing the dynamics rank to the true underlying dimensionality of the state manifold (in this case, 2) allows for more expressive dynamics and avoids the drastic drop in consistency as training progresses. Furthermore, the Jacobian and cost loss terms do not substantially degrade performance on the primary reconstruction and dynamics learning tasks.

We also implemented an iLQR variant using our proposed model to provide the dynamics and cost function. Unfortunately, due to time constraints we were only able to conduct limited comparison of its performance relative to other controllers. We did perform an experiment on the pendulum task and found that our model, with all additional loss terms, is able to stabilize an inverted pendulum, as shown in Figure 5. The baseline model, which used only a quadratic cost from the target in latent space, was unable to swing up the pendulum.

Unfortunately, the learned model is very fragile and sensitive to the parameters of the iLQR algorithm, and changing the horizon or regularization parameters of the iLQR optimizer can break it. Because iLQR is so sensitive to the accuracy of the learned Jacobians, it is very easy for the optimizer to get "stuck" and be unable to make progress when the approximate Jacobians are misaligned with the true values, which can happen particularly often when out-of-distribution latent states are reached.

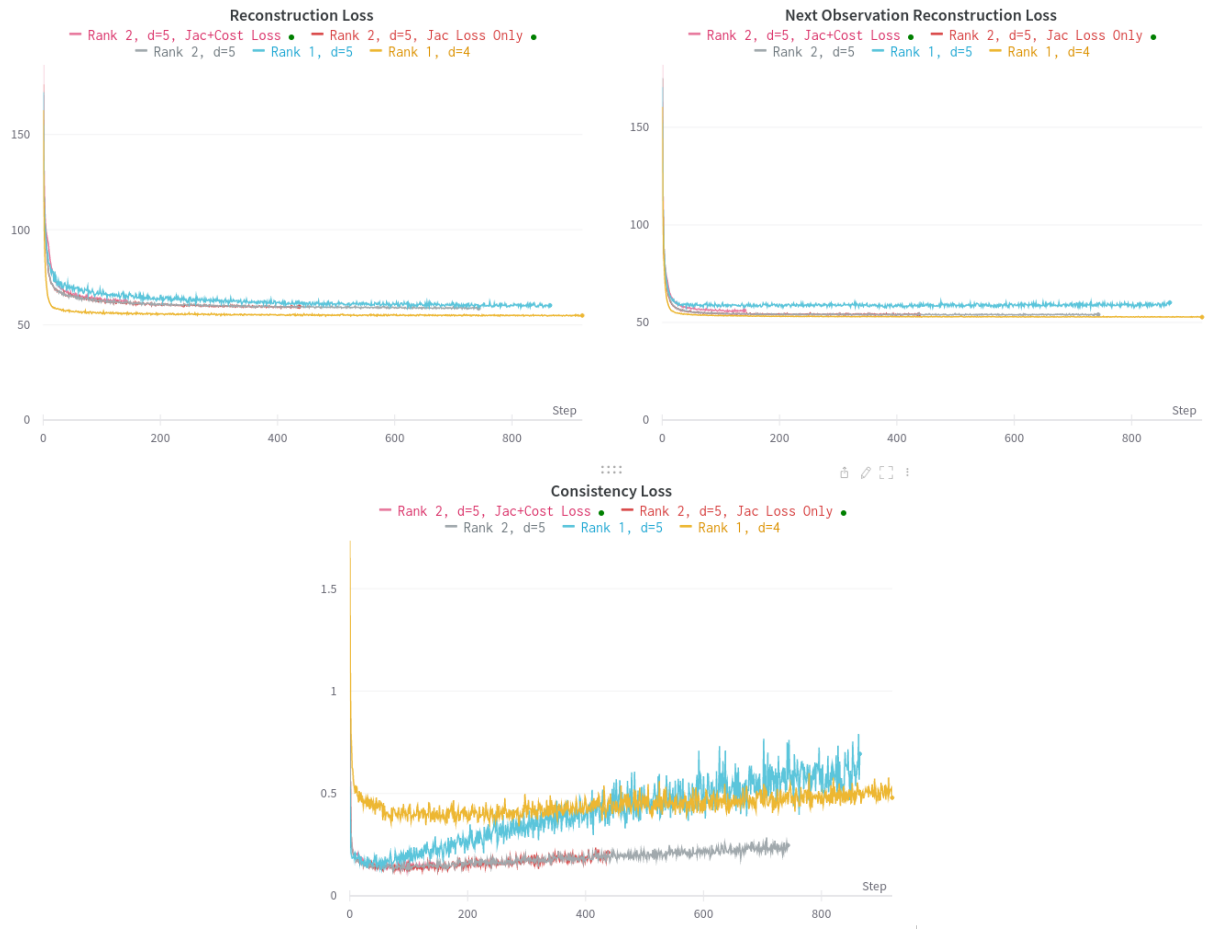

Figure 3: Training loss curves for dynamics. Top left: reconstruction loss curve

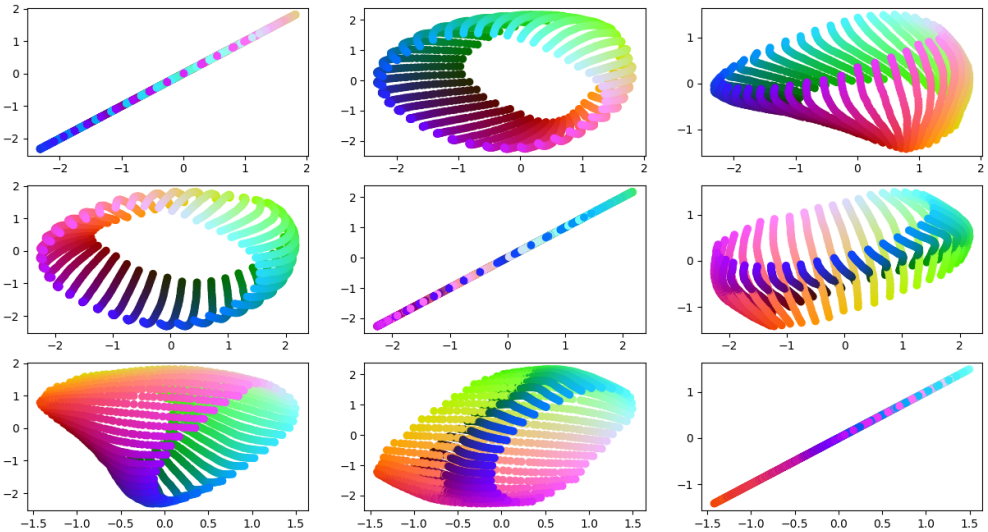

Figure 4: Slices of first three dimensions of pendulum latent space embeddings in $n = 5$ dimensions. The cylindrical structure of the state manifold is apparent.

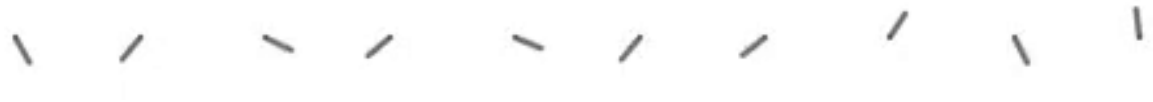

Figure 5: Trace of pendulum using iLQR and latent-space dynamics and costs, with our added terms. With lower latent dimension the learning process failed to discover a reasonable latent state, and without the addition of rank-2 the pendulum was unable to swing up.

VI Conclusion

In this paper, we discussed several existing algorithms for model-based control, with a focus on algorithms that plan using a latent representation of the state. In particular, we considered the embed-to-control algorithm and discussed several shortcomings of its proposed approach. We analyze these shortcomings theoretically and propose modifications that should address each of them. Finally, we implement a modified version of this algorithm with our proposed changes and demonstrate that it is able to recover a good latent state representation and can successfully carry out a very simple visuomotor control task.

While we were able to demonstrate a proof-of-concept of our modifications to E2C and demonstrate that it avoids several common failure cases of the original embed-to-control framework, we were not able to conduct rigorous quantitative experiments comparing our algorithm to other model-based RL algorithms. In future work, we would like to continue this analysis and conduct experiments in nontrivial environments. In particular, there are several MuJoCo tasks that could prove informative; it would be interesting to see if the latent dynamics model can learn effective smooth approximations to non-smooth contact dynamics.

Finally, a major disadvantage of model-based reinforcement learning methods in general is that they are greatly succeptible to distribution shift: the dataset must cover not only states that are visited, but all states that could possibly be sampled by the optimizer. This proves impossible in high-dimensional latent spaces in which the vast majority of possible latent states do not even correspond to a real state. It may be possible to use our residual cost-learning framework alongside some sort of contrastive loss to ensure that latent states that do not correspond to a real state in the dataset are assigned high cost.

Our project is in the GitHub repository [7]. Our code makes use of a third-party reimplementation of the dynamics network models from the original E2C paper [8].

Acknowledgement

Kyle Stachowicz worked on the formulation of the problem, implementation of our method with new loss functions, and provided a solution to the latent dimension issue as discussed in Section A.

Kai Yun worked on the formulation of the problem, implementation of offline data collection, and training infrastructure.

Haoming Guo worked on the implementation of convolutional neural networks (CNN) for image processing and on training infrastructure.

References

[1] M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *ArXiv:1506.07365v3*, 2015.

[2] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *ArXiv:1312.6114*, 2013.

[3] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *ArXiv:1805.12114v2*, 2018.

[4] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting based on approximating dynamic programming. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[5] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *International Conference on Informatics in Control, Automation and Robotics*, 2004.

[6] Sergey Levine. CS 285, Lecture 10: Optimal Control and Planning. *University of California, Berkeley*, 2022.

[7] Haoming Guo Kyle Stachowicz, Kai Yun. cs285_project. `https://github.com/kaiyun717/cs285_project`, 2022.

[8] Tung Nguyen. E2C-pytorch. `https://github.com/tung-nd/E2C-pytorch`, 2019.