2021 18th Conference on Robots and Vision (CRV) | 978-1-6654-1413-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/CRV52889.2021.00029

Deep Koopman Representation for Control over Images (DKRCI)

Philippe Laferrière*†, Samuel Laferrière‡, Steven Dahdah§, James Richard Forbes§¶, and Liam Paull*†
* *Department of Computer Science and Operations Research, Université de Montréal, Montreal, Quebec,*
`philippe.laferriere.1@umontreal.ca`
† *Mila, Montreal, Quebec*
‡ *Orangead Media, Montreal, Quebec*
§ *Department of Mechanical Engineering, McGill University, Montreal, Quebec*
¶ *GERAD, HEC Montreal, Montreal, Quebec*

***Abstract*—The Koopman operator provides a means to represent nonlinear systems as infinite dimensional linear systems in a lifted state space. This enables the application of linear control techniques to nonlinear systems. However, the choice of a finite number of lifting functions, or Koopman observables, is still an unresolved problem. Deep learning techniques have recently been used to jointly learn these lifting function along with the Koopman operator. However, these methods require knowledge of the system's state space. In this paper, we present a method to learn a Koopman representation directly from images and control inputs. We then demonstrate our deep learning architecture on a cart-pole system with external inputs.**

***Keywords*-Koopman operator theory, deep learning, dynamical systems**

I. Introduction

A discrete-time initial value problem (IVP) consists of a dynamical system

$$x_{k+1} = f(x_k), \tag{1}$$

along with an initial value x_0. Solving an IVP yields a trajectory $(x_0, x_1, \ldots, x_n)$, where n may be as large as desired. Certain problems add *control inputs* to this formulation, which must be be individually set at each time step. Then, the dynamical system formulation becomes

$$x_{k+1} = f(x_k, u_k), \tag{2}$$

where u_k is the control input.

Three distinct problems arise from such a formulation [1]. The first is the *system identification* problem, in which we wish to identify f from a dataset of (x_k, u_k, x_{k+1}) tuples. The second is the *simulation* or *prediction* problem, in which we wish to find x_{k+1} given $f(\cdot,\cdot)$ and (x_k, u_k) pairs. This problem is particularly relevant when we have a good model of a system and want to predict its behavior under certain circumstances. The third problem is the *control* problem, wherein we want to find a *control law* or *policy*

$$u_k = h(x_k) \tag{3}$$

that will drive the system to a desired state while starting from some initial state x_0.

In this paper, we focus on the system identification and simulation problems. Traditionally, when studying a system of interest, expert knowledge is needed to formulate and derive a parametric differential or difference equation that can reasonably model the observed data. The system identification problem then amounts to identifying, or learning, the parameters of these equations [2]. Nowadays, due to advances surrounding deep learning [3], it is common to model f using a neural network [4], [5]. The system identification problem then translates to training that neural network using data gathered from running the dynamical system from various initial conditions. This is sometimes referred to as model-based learning, as opposed to the model-free setting [6]. The control problem, on the other hand, is more challenging. While control theory for linear dynamical systems is generally well understood, controlling arbitrary nonlinear dynamical systems is still challenging.

The Koopman operator [7], [8] is an alternative approach to representing nonlinear dynamics, wherein nonlinear dynamics are lifted to an infinite dimensional space where they appear linear. Thus, the Koopman operator trades finite-dimensional nonlinear dynamics for infinite dimensional linear dynamics. The exact representation of a nonlinear system as a linear system is of great consequence for control design, as linear control techniques may be used in the lifted space, which correspond to nonlinear controllers in the original state space. Using the Koopman operator, linear control techniques like model predictive control have been successfully applied to to nonlinear systems [9].

For practical use, a finite dimensional approximation of the Koopman operator must be constructed. To do so, a finite selection of *lifting functions* or *observables* must be selected. For that given choice of observables, an approximate Koopman operator is then found using regression techniques. The choice of Koopman observables is an open research problem. They are often hand-engineered based on known dynamics [10], [11], or selected to be standard basis functions like polynomials [12].

Deep learning techniques have proven effective in jointly selecting Koopman observables and approximating the Koopman operator from data [13], [10], [14], particularly for systems where the state space is readily accessible. Lusch *et al.* [13] use a deep neural network to represent their Koopman observables, and jointly optimize for the neural network parameters and Koopman operator. Abraham and Murphey [10] go a step further by considering control inputs and synthesizing a linear controller using the learned Koopman operator. They then employ an active learning loop to stabilize a cart-pole system and control a two-link robot arm. Han *et al.* [14] present a similar learning approach with a modified loss function that promotes controllability in the learned Koopman operator. They then demonstrate their approach, called Deep Koopman Representation for Control (DKRC), using inverted pendulum and lunar lander simulations from OpenAI Gym [15].

The above approaches require access to the state space of each system, which is not always available in robotics problems. The contribution of our work is to extend DKRC to the image domain. We first use an autoencoder to translate images into a lower dimensional latent space. We then treat that latent space as the state space and use a DKRC-based approach to jointly learn the Koopman operator and observables. We call this approach DKRCI: *Deep Koopman Representation for Control over Images*. While this paper was in review, the authors noticed that Xiao *et al.* [16] proposed a similar approach to ours, which they call CKNet. However, the major difference between the two architectures is that we divide the architecture in two stages to faciliate training (see section III for details), whereas they map from images directly to observation space.

II. Background

Consider a discrete-time nonlinear ODE

$$x_{k+1} = f(x_k) \tag{4}$$

with initial condition x_0, where $x_k \in \mathbb{R}^{n_x}$. The *Koopman observables* $g : \mathbb{R}^{n_x} \to \mathbb{R}$ form an infinite dimensional vector space $\mathbb{F}$. The evaluation of an observable at a point is called an *observation*. The discrete-time *Koopman operator* $\mathbb{K} : \mathbb{F} \to \mathbb{F}$ advances observables in time. It is defined as

$$\mathbb{K}(g) = g \circ f \tag{5}$$

for any $g \in \mathbb{F}$. Evaluated at x_k, that is

$$\mathbb{K}(g)(x_k) = (g \circ f)(x_k) = g(x_{k+1}). \tag{6}$$

The Koopman operator returns the observable that, when evaluated at the current state, maps to the observations of the next time step. The Koopman operator advances *functions* through time as opposed to advancing states.

When a vector space structure is defined on $\mathbb{F}$, the Koopman operator is linear, meaning

$$\begin{aligned}(\mathbb{K}(\lambda_1 g_1 + \lambda_2 g_2))(x) &= (\lambda_1 g_1 + \lambda_2 g_2)(f(x)) && (7)\\ &= \lambda_1 g_1(f(x)) + \lambda_2 g_2(f(x)) && (8)\\ &= \lambda_1 (\mathbb{K} g_1)(x) + \lambda_2 (\mathbb{K} g_2)(x). && (9)\end{aligned}$$

As a result, the attractive properties of superposition and scaling hold.

Therein lies the tradeoff made by Koopman operator theory: trade a finite dimensional but nonlinear dynamical system for an equivalent linear but infinite dimensional dynamical system. The goal of applied Koopman operator theory is to find a good approximation of the Koopman operator on some space of functions.

In practice, we must limit ourselves to a finite-dimensional space of functions $\mathcal{F} \subset \mathbb{F}$, so $\mathcal{K}$, the approximation of $\mathbb{K}$ on $\mathcal{F}$, can be represented as a matrix. We define a dictionary of observables $D_x = [\phi_1 \ \cdots \ \phi_n]^\top$ which spans $\mathcal{F}$. Then, for all $g \in \mathcal{F}$, $g = \sum_{i=1}^n w_i \phi_i$, $w_i \in \mathbb{R}$. Put another way, we can represent every function g in $\mathcal{F}$ as a linear combination of the elements of D_x, and write $g = [w_1 \ \cdots \ w_n]^\top$. Note that D_x is not necessarily a basis for $\mathcal{F}$; some vectors may be linear combinations of others. For example, D_x could be all polynomials of the state vector up to degree p. However, the choice of D_x is crucial and makes or breaks the performance of the algorithm. From a theoretical point of view, a good choice of dictionary is one in which the resulting $\mathcal{F}$ is an invariant subspace under $\mathbb{K}$, or as much as possible. From a practical point of view, as the dimension of D_x increases, we have found that an increasing amount of numerical issues arise when learning $\mathcal{K}$.

Using this representation for the functions of $\mathcal{F}$, $\mathcal{K}$ is an $n \times n$ matrix, and

$$g(x_{k+1}) = (\mathcal{K} g)(x_k) + e_k$$

where $e_k \in \mathbb{F}$ is the error that arises from using $\mathcal{K}$ instead of $\mathbb{K}$. Another interpretation is that $\mathcal{K} = \mathbb{P} \circ \mathbb{K}$, where $\mathbb{P}$ is an operator which projects vectors $g \in \mathbb{F}$ on $\mathcal{F}$.

Not only can the Koopman framework be used for prediction, it can also be used in control. By using a linear representation of the dynamical system, as opposed to a nonlinear one, linear control tools such as the linear-quadratic regulator (LQR) [10], [14] or model predictive control [9], [17] can be used.

The framework presented so far can be adapted to include an exogenous inputs u_k. Consider a dynamical system $x_{k+1} = f(x_k, u_k)$, and let $D = [D_x, D_u] = [\phi_1 \ldots \phi_{n_x}, \psi_1, \ldots, \psi_{n_u}]^\top$, where $\phi_i(x) : \mathbb{R}^{n_x} \to \mathbb{R}$ and $\psi_i(x, u) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$. We separate the dictionary in such a way because we do not want to predict the control variables; these are chosen by a controller outside the system. Thus, $\mathcal{K}$ can be written as [10]

$$\mathcal{K} = \begin{bmatrix} \mathcal{K}_x & \mathcal{K}_u \\ \cdot & \cdot \end{bmatrix}, \tag{10}$$

$\mathcal{K}_x \in \mathbb{R}^{n_x \times n_x}$ and $\mathcal{K}_u \in \mathbb{R}^{n_x \times n_u}$. The $(\cdot)$ in (10) refers to the terms that evolve the control inputs, which we don't estimate. The dynamical system over the observables becomes

$$D_x(x_{k+1}) = \mathcal{K}_x D_x(x_k) + \mathcal{K}_u D_u(x_k, u_k). \tag{11}$$

III. Learning with Koopman

A simple and efficient way to learn the Koopman operator is by a simple least-squares optimization problem [18]. Thus, given a dataset $(x_i, u_i, y_i)_{i=1}^m$, where $y_i = f(x_i, u_i)$,

$$\mathcal{K} = \arg\min_K \frac{1}{2}\sum_{i=1}^m \left\|D_x(y_i) - K_x^\top D_x(x_i) - K_u^\top D_u(x_i, u_i)\right\|_2^2 + \lambda \|K\|_{\mathrm{r}}^2 \tag{12}$$

where $D_x(x) = [\phi_1(x) \ldots \phi_n(x)]$ and the regularizer norm is $\mathrm{r} = 1$ or $\mathrm{r} = 2$. Notice that we are doing the regression in the space of *observations* as opposed to the space of *observables*, while the Koopman operator acts on observables as opposed to observations. It can be shown that the operator K that acts on the space of observations is the transpose of $\mathcal{K}$. Hence we write $K^\top$ as opposed to K.

One must be careful when choosing D_u. When doing control, observables that predict well may not be suited for control. We make this more explicit with an example. We present a continuous dynamical system that can naturally be translated into a discrete-time dynamical system.

Let our state space be $\mathbf{x} = [x\ y\ \theta]^\top$ with control $\mathbf{u} = [v\ \omega]^\top$. The transition function $f(\mathbf{x}, \mathbf{u})$ is

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}. \tag{13}$$

Choose

$$D_x(\mathbf{x}) = \begin{bmatrix} x \\ y \\ \sin\theta \\ \cos\theta \end{bmatrix}, \quad D_u(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega\cos\theta \\ \omega\sin\theta \end{bmatrix}. \tag{14}$$

Given this choice of observables, there exists a finite-dimensional Koopman operator that gives perfect predictions, that being

$$\frac{\mathrm{d}}{\mathrm{d}t} D_x(\mathbf{x}) = \mathbb{K}_x D_x(\mathbf{x}) + \mathbb{K}_u D_u(\mathbf{x}, \mathbf{u}) \tag{15}$$

$$= \mathbf{0}\, D_x(\mathbf{x}) + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} D_u(\mathbf{x}, \mathbf{u}). \tag{16}$$

It can be shown that these equations are exact. Now, for the controller, we will use LQR with $\mathbf{Q} = \mathbf{I}$ and $\mathbf{R} = \mathbf{I}$. Therefore, our control $D_u(\mathbf{x}, \mathbf{u})$ has the form $D_u(\mathbf{x}, \mathbf{u}) = -\mathbf{F}_{\text{ctrl}} D_x(\mathbf{x})$. It turns out that

$$\mathbf{F}_{\text{ctrl}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

such that

$$D_u(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega\cos\theta \\ \omega\sin\theta \end{bmatrix} = \begin{bmatrix} -x \\ -y \\ \sin\theta \\ -\cos\theta \end{bmatrix}. \tag{18}$$

There are multiple ways to recover v and ω. For example,

$$v = \frac{-x}{\cos\theta}, \tag{19}$$

$$v = \frac{-y}{\sin\theta}. \tag{20}$$

What's concerning is the fact that using (19) or (20) results in a different v. In particular, $D_u(\mathbf{x}, \mathbf{u})$ lies on a manifold, whereas $-\mathbf{F}_{\text{ctrl}} D_x(\mathbf{x})$ lies in a vector space. The various ways of recovering v and ω can be thought of as choosing different projections back onto the manifold. This is a potential shortcoming of applying Koopman theory when inputs are present.

Returning to this specific example, consider the following ω and v control inputs derived directly from (18):

$$\omega = \sqrt{\omega^2\cos^2\theta + \omega^2\sin^2\theta} \tag{21}$$

$$= \sqrt{\sin^2\theta + (-\cos\theta)^2} \tag{22}$$

$$= 1, \tag{23}$$

and

$$v = \sqrt{v^2\cos^2\theta + v^2\cos^2\theta} \tag{24}$$

$$= \sqrt{x^2 + y^2}. \tag{25}$$

This is clearly a bad controller; $\omega = 1$ corresponds to a constant anti-clockwise input, while the tangential velocity is larger the farther the system is from the origin. It can be verified that other vs and ωs derived from (18) give equally bad controllers. This motivated us to pick $D(\mathbf{x}, \mathbf{u}) = \mathbf{u}$ so that we avoid any ambiguities in recovering $\mathbf{u}$.

A. DKRCI

In virtually every robotics application, we do not have access to the underlying state space. This calls for a method which acts on sensor data directly. Next we present our method, DKRCI, which extends DKRC to the image domain.

$$\begin{tikzcd}
(\mathbb{Z}, \mathbb{W}) \arrow[r, "K"] & \mathbb{Z} \arrow[d, "\phi_r"] \\
\mathbb{Y} \arrow[u, "\phi_f"] & \mathbb{Y}
\end{tikzcd}$$

Figure 1: DKRC architecture. $\mathbb{Y}$ is the state space, $\mathbb{Z}$ is the observation space, and $\mathbb{W}$ is the action space. K is the Koopman operator.

Algorithm 1: DKRCI Training Process

```
Result: Trained autoencoder and Koopman network
// Train autoencoder
for number of training iterations do
    Sample minibatch of $m$ images $\{x_k\}_{k=1}^{m}$ from
      the autoencoder dataset ;
    Update autoencoder parameters $\theta_a$ using Adam
      optimizer;
        $\nabla_{\theta_a} \sum_{i=1}^{m} \left\| x_i - \varphi_r \circ \varphi_f(x_i) \right\|_1$
end
// Train Koopman network
for number of training iterations do
    Sample minibatch of $n$ data points $\{(x_k, u_k)\}_{k=1}^{n}$
      from the Koopman network dataset ;
    Update Koopman network parameters $\theta_k$ using
      Adam optimizer;
        $\nabla_{\theta_a} \sum_{k=1}^{n} \gamma^t \left\| x_k - \varphi_r \circ \phi_r \circ K^t(\phi_f \circ \varphi_f(x_1), u_{1:t}) \right\|$
end
```

1) Architecture Overview: The original DKRC architecture is shown in Figure 1 and our proposed architecture is shown in Figure 2. We define four spaces: *pixel* space $\mathbb{X}$, *latent* space $\mathbb{Y}$, *observation* space $\mathbb{Z}$, and *action* space $\mathbb{W}$. It is designed as a two-tiered system: an autoencoder (φ_f, φ_r) which relates the pixel space $\mathbb{X}$ and the latent space $\mathbb{Y}$, and a Koopman network (ϕ_f, K, ϕ_r) which relates the latent space $\mathbb{Y}$ and the observation space $\mathbb{Z}$. This design allows us to decouple the training of the system into two separate phases, which greatly simplifies the process. Additionally, note that K takes an input variable $w \in \mathbb{W}$ directly, as opposed to sending ω through $\phi_f \circ \varphi_f$. The reason for this is explained in Section II.

2) Datasets: In order to train DKRCI, we need to collect two different datasets. The first dataset, the autoencoder dataset, is a collection of images collected simply by running the environment with random inputs. The second dataset, the Koopman network dataset, is a collection of episodes $\{(x_k, u_k)\}_{k=1}^{n}$ of length n, where x_k is the image at time k, and u_k is the random control applied at time k. In order for the model to generalize well, care must be taken to gather data from as large a subset of pixel space as possible.

3) Training process: The training process is depicted in Algorithm 1. The first training phase consists of training the autoencoder, which consists of the encoder φ_f and the decoder φ_r. These are trained jointly over a dataset of images using the Adam optimizer [19].

The second phase is more involved. During this training phase, the autoencoder is held constant, and the three parts of the Koopman network are jointly trained again with the Adam optimizer: the observables network ϕ_f, the Koopman operator K, and the recover network ϕ_r. The challenge is to ensure that what the K network learns is actually the Koopman operator of the underlying dynamical system, which we enforce with a soft constraint. We train this second phase using multi-step prediction, where K is applied n times to the observation given by the observables network, and each of these predictions has to be recovered by the recover network. More specifically, the loss function is given by

$$L = \sum_{k=1}^{n} \gamma^k \left\| x_k - \varphi_r \circ \phi_r \circ K^k(\phi_f \circ \varphi_f(x_1), u_{1:t}) \right\| \quad (26)$$

where $\gamma \in [0, 1]$ is the discount factor, x_k are the images, and u_k are the input. This loss function highlights a crucial difference between deep Koopman approaches and other deep approaches to representing dynamics with deep neural networks [5]: moving one extra step into the future only requires one extra matrix multiplication of K. Other approaches typically require a full neural network feedforward pass in order to move one extra step into the future.

IV. Results

Similar to DKRC [14], we use a modified version of the cart-pole environment of OpenAI Gym [15] to test our algorithm. The default cart-pole environment only accepts two possible discrete actions: 0 to nudge the cart-pole left, and 1 to nudge it right. However, the Koopman operator is formulated for differential equations with states $x \in \mathbb{R}^{n_x}$. We used a modified cart-pole environment that accepts all values between -1 and 1 as input, where -1 corresponds to maximum force applied to the left, while 1 corresponds to maximum force to the right. We sampled datasets of trajectories from this environment, and used them to train our neural networks in two distinct phases as described in Section III.

In order to find the best neural network architecture for the autoencoder and the Koopman network, we varied the number of layers (between 2 and 3), the activation functions

$$\begin{tikzcd}
(\mathbb{Z},\mathbb{W}) \arrow[rr, "K"] & & \mathbb{Z} \arrow[d, "\phi_r"] \\
\mathbb{Y} \arrow[u, "\phi_f"] & & \mathbb{Y} \arrow[d, "\varphi_r"] \\
\mathbb{X} \arrow[u, "\varphi_f"] & & \mathbb{X}
\end{tikzcd}$$

Figure 2: DKRCI architecture. An image at time k is mapped to latent space with the encoder φ_f, which is mapped into observation space with neural network ϕ_f. The action $w \in \mathbb{W}$ is appended to the observation and mapped back into observation space using the Koopman operator K. This represents the observation at time $k+1$. This observation is mapped back into latent space using ϕ_r, and back into pixel space using φ_r, which represents the image at time $k+1$.

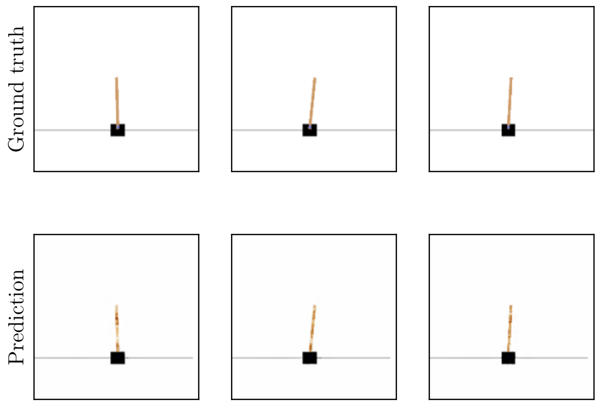


Figure 3: Comparison between the real image and the autoencoder's prediction. The first row shows the real images; the second shows the autoencoder's prediction.

(ReLU, Leaky ReLU, or tanh), the use or not of batch normalization [20] and the number of data points used in training (1×10^3, 1×10^4, or 5×10^4).

The final architecture is depicted in Figure 2. In our implementation, $\mathbb{X} = [0,1]^{210\times210}$, $\mathbb{Y} = \mathbb{R}^{10}$, $\mathbb{Z} = \mathbb{R}^{31}$, and $\mathbb{W} = [-1,1]$. An autoencoder, composed of encoder φ_f and decoder φ_r, is trained to map to and from image and latent spaces. A neural network ϕ_f maps from latent space to observation. Similarly, a second neural network ϕ_r maps from observation space back into latent space. Finally, the Koopman operator is a linear operator that maps from observation space (cross action space) to observation space.

A. Autoencoder architecture

The images that go into the autoencoder are resized to 210×210 pixels using bilinear interpolation, and pixel values are mapped to the $[0,1]$ range. The autoencoder can be thought of as two separate networks: the encoder φ_f, and the decoder φ_r.

The encoder φ_f is a convolutional neural network consisting of two convolutional layers, each feeding into a batch norm layer [20], followed by a ReLU nonlinearity and 10×10 max pooling. The first convolutional layer is an 11×11 kernel and maps to 64 output channels. The second convolutional layer also uses an 11×11 kernel, mapping to 256 output channels. Finally, the last layer is a linear function mapping to 10 output channels (the latent space $\mathbb{Y}$), followed by a leaky ReLU layer.

The decoder φ_r is comprised of two deconvolutional layers, each feeding into a batchnorm layer. First, the 10-dimensional input is sent through a linear layer that maps to 256 dimensions, followed by a leaky ReLU. Then, 10×10 max unpooling is applied, followed by a ReLU nonlinearity, followed by a batch norm layer, and finally through the first deconvolutional layer (11×11 kernel, mapping 256 channels to 64). The output is again fed into a 10×10 max unpooling layer, into a ReLU nonlinearity, into a batchnorm layer, and finally into a deconvolutional layer (11×11 kernel, mapping 64 channels to 3). To ensure that φ_r maps into $[0,1]$, we send this output through a batchnorm layer, followed by a sigmoid nonlinearity.

B. Observable network ϕ_f's architecture

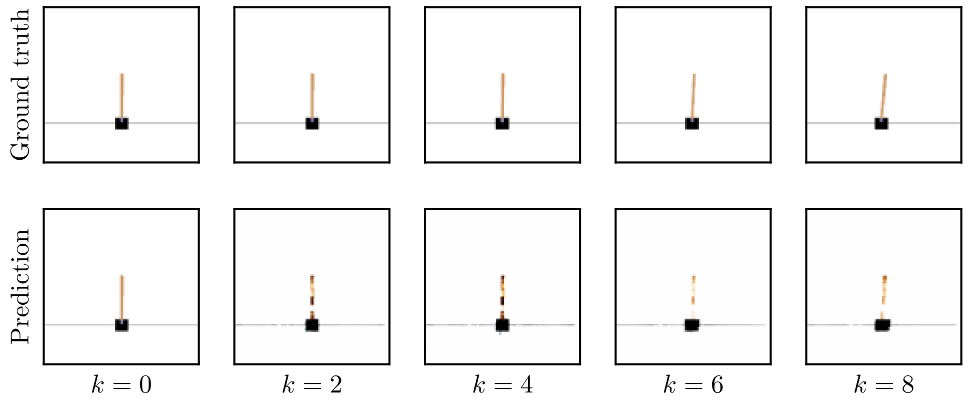
ϕ_f is a two-layered multi-layered perceptron (MLP). The first layer maps from 10 dimensions to 512, followed by a leaky ReLU nonlinearity. The second layer maps from 512 dimensions to 31 dimensions (dimensionality of $\mathbb{Z}$ in our implementation), followed by a leaky ReLU nonlinearity.

C. Recover network ϕ_r's architecture

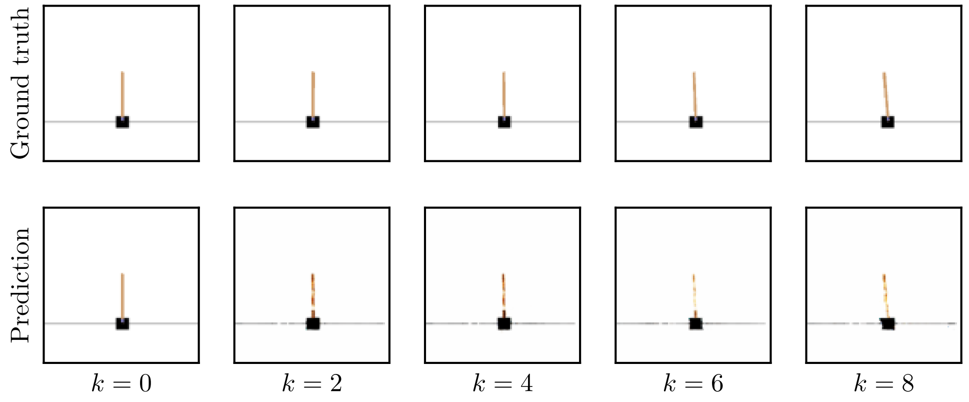
ϕ_r is similar to ϕ_f. It is a two-layered MLP. The first layer maps from 31 dimensions to 512, followed by a leaky ReLU nonlinearity. The second layer maps from 512 dimensions to 10 dimensions (dimensionality of $\mathbb{Y}$ in our implementation), followed by a leaky ReLU nonlinearity.

First, we train the autoencoder. To build our dataset of images, we ran the cart-pole OpenAI gym environment using random actions until we had 10 000 images collected. We used an L_1 loss over pixels where the values are mapped to the $[0,1]$ range. We trained for 50 000 iterations. The results are shown in Figure 3.

In order to verify that multi-step prediction works well, we use a test data set where for each episode, the same action is applied 10 times. This allows the cart-pole stick to swing completely in one direction, and thus we could see that the system had learned how actions changed the dynamical system. The results are shown in Figure 4. What these results



(a) Constant input of 0 (*i.e.*, "move left").



(b) Constant input of 1 (*i.e.*, "move right").

Figure 4: Simulation and prediction of a cart-pole system with the same action is applied at every step. In each row, the same input is applied for the whole episode. The DKRCI predictions agree with ground truth.

show is that we were able to train a model which can predict future states in pixel space given a series of states and actions applied using a Koopman representation. In other words, we have appropriately learned a linear representation of our nonlinear dynamical system entirely from data.

V. Conclusion

The Koopman operator, at its essence, trades a finite dimensional nonlinear dynamical system for a linear but infinite dimensional equivalent system. The linear nature of the Koopman operator, and its approximation, is attractive from both prediction and control perspectives. In this paper we explored the DKRC framework, which combines the power of deep neural networks with Koopman operator based methods to synthesize a controller over the ground truth state space. We showed how to extend this into the image domain, as typically one doesn't have access to the underlying state space. Future work involves extending our method to do control over this image domain. Specifically, it is currently not clear how to define the *error* dynamics so that the origin refers to a desired state, such as the cart-pole being upright in the case of the cart-pole environment. When one does this on the underlying state space (lateral position d and cart-pole angle θ), one can use LQR to drive the state to $\theta = 0$ independent of the value d. It is difficult to do that in image, latent or observation space because we don't have an explicit representation of d and θ.

Acknowledgment

This research was supported by the CIFAR Pan-Canadian AI Strategy through a Catalyst Grant and Mitacs Accelerate Fellowship.

References

[1] B. Douglas, *An Engineer's Guide to The Fundamentals of Control Theory*, 2019, self-published e-book.

[2] L. Ljung, *System Identification: Theory for the User*. PTR Prentice Hall, 1999.

[3] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, 2015.

[4] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, “Information theoretic MPC for model-based reinforcement learning,” in *2017 Int. Conf. Robotics and Automation*, 2017, pp. 1714–1721.

[5] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, “Learning latent dynamics for planning from pixels,” in *Proc. 36th Int. Conf. Machine Learning*, ser. Proc. Machine Learning Research, vol. 97. PMLR, 2019, pp. 2555–2565.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[7] B. O. Koopman, “Hamiltonian systems and transformations in Hilbert space,” *Proc. National Academy of Sciences*, vol. 17, no. 5, pp. 315–318, 1931.

[8] A. Mauroy, I. Mezić, and Y. Susuki, *The Koopman Operator in Systems and Control*. Springer International, 2020.

[9] M. Korda and I. Mezić, “Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control,” *Automatica*, vol. 93, pp. 149–160, 2018.

[10] I. Abraham and T. D. Murphey, “Active learning of dynamics for data-driven control using Koopman operators,” *IEEE Trans. Robot.*, vol. 35, no. 5, pp. 1071–1083, 2019.

[11] G. Mamakoukas, M. Castano, X. Tan, and T. Murphey, “Local Koopman operators for data-driven control of robotic systems,” in *Proc. Robotics: Science and Systems XV*, 2019.

[12] D. Bruder, C. D. Remy, and R. Vasudevan, “Nonlinear system identification of soft robot dynamics using Koopman operator theory,” `arXiv:1810.06637 [cs.RO]`, 2019.

[13] B. Lusch, J. N. Kutz, and S. L. Brunton, “Deep learning for universal linear embeddings of nonlinear dynamics,” *Nature Communications*, vol. 9, no. 1, pp. 1–10, 2018.

[14] Y. Han, W. Hao, and U. Vaidya, “Deep learning of Koopman representation for control,” in *59th IEEE Conf. Decision Control*, 2020, pp. 1890–1895.

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” `arXiv:1606.01540 [cs.LG]`, 2016.

[16] Y. Xiao, X. Xu, and Q. Lin, “CKNet: A convolutional neural network based on Koopman operator for modeling latent dynamics from pixels,” `arXiv:2102.10205 [eess.SY]`, 2021.

[17] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan, “Modeling and control of soft robots using the Koopman operator and model predictive control,” in *Proc. Robotics: Science and Systems XV*, 2019.

[18] I. Abraham and T. D. Murphey, “Active learning of dynamics for data-driven control using Koopman operators,” *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1071–1083, 2019.

[19] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” `arXiv:1412.6980 [cs.LG]`, 2014.

[20] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” `arXiv:1502.03167 [cs.LG]`.