ANALYSIS AND REGULARIZATION OF DEEP GENERATIVE SECOND ORDER ORDINARY DIFFERENTIAL EQUATIONS

by

Batuhan Koyuncu

B.S., Physics, Boğaziçi University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2021

ACKNOWLEDGEMENTS

I would like to express my gratitude towards my advisor Prof. Lale Akarun for allowing me to pursue my master's degree under her supervision and guidance. I also would like to thank Prof. Ali Taylan Cemgil for accepting me as a graduate student prior to his leave. His lectures and approach have inspired me to pursue my degree in machine learning. I would also like to thank my thesis jury members Prof. Mehmet Burçin Ünlü and Assist. Prof. Berk Gökberk for their valuable comments and suggestions. In addition, I thank Prof. Ünlü for all his guidance and encouragement through the years.

I owe my deepest gratitude to my family for their endless support, encouragement, and love, especially during difficult times. I thank my dearest friends Alp, Gökçe, Miray, Mustafa, Onur, and Orhan. I feel blessed to have their support and friendship. I would like to thank Ekin for offering her endless support and always being there for me. She has turned this journey into a joyful and inspirational one.

I would like to thank my colleagues at Boğaziçi University for their support and friendship. I thank Ahmet, Ahmet Alp, Doğa, Oğulcan,and Tuna Han for offering their help whenever I need it.

This work is supported by Boğaziçi University Research Fund under the Grant Number 16903. We thank TETAM and Inzva for the computing resources provided.

ABSTRACT

ANALYSIS AND REGULARIZATION OF DEEP GENERATIVE SECOND ORDER ORDINARY DIFFERENTIAL EQUATIONS

Deep generative models aim to learn processes that are assumed to generate the data. To this end, deep latent variable models use probabilistic frameworks to learn a joint probability distribution over the data and its low-dimensional hidden variables. A challenging task for the deep generative models is learning complex probability distributions over sequential data in an unsupervised setting. Ordinary Differential Equation Variational Auto-Encoder (ODE2VAE) is a deep generative model that aims to learn complex generative distributions of high-dimensional sequential data. The ODE2VAE model uses variational auto-encoders (VAEs) and neural ordinary differential equations (Neural ODEs) to model low-dimensional latent representations and continuous latent dynamics of the representations, respectively. In this thesis, we aim to explore the effects of the inductive bias in the ODE2VAE model by analyzing the learned dynamic latent representations over three different physical motion datasets. Then, we re-formulate the model for flexible regularization, and we extend the model architecture to facilitate the learning of the varying static features in the sequential data. Through the experiments, we uncover the effects of the inductive bias of the ODE2VAE model over the learned dynamical representations and demonstrate the ODE2VAE model's shortcomings when it is used for modeling sequences with varying static features.

ÖZET

DERİN ÜRETİCİ İKİNCİ DERECEDEN ADİ DİFERANSİYEL DENKLEMLERİN ANALİZİ VE DÜZENLENMESİ

Derin üretici modeller, verileri ürettiği varsayılan süreçleri öğrenmeyi amaçlar. Bu amaçla, derin üretici modeller, veriler ve verilerin düşük boyutlu gizli değişkenleri üzerindeki ortak olasılık dağılımını modeller. Sıralı veriler üzerindeki karmaşık olasılık dağılımlarını gözetimsiz biçimde öğrenmek derin üretici modeller için zorlu bir görevdir. Adi Diferansiyel Denklem Değişimsel Otokodlayıcı (ADDDO) yüksek boyutlu sıralı verilerin kompleks üretici dağılımlarını öğrenmeyi amaçlayan bir derin üretici modeldir. ADDDO modeli düşük boyutlu gizli değişkenleri ve değişkenlerin sürekli gizli dinamiklerini modellemek için sırasıyla değişimsel otokodlayıcı ve sinirsel adi diferansiyel denklem modellerini kullanır. Bu tezde, üç farklı fiziksel hareket veri setinde öğrenilen dinamik gizli temsilleri analiz ederek ADDDO modelinin sahip olduğu model varsayımının etkilerini incelenmiştir. Ardından verilerdeki farklılaşan statik özelliklerin öğrenilmesini kolaylaştırmak amacıyla bu model esnek düzenlileştirilme için yeniden formüle edilmiş ve ayrıca model mimarisi genişletilmiştir. Deneyler sonucunda ADDDO modelinin model varsayımının öğrenilen dinamik gösterimler üzerindeki etkileri ortaya çıkarılmış ve bu modelin, değişken statik özelliklere sahip dizileri modellemek için kullanıldığında yetersiz kaldığı gösterilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a	Latent dimensionality
$\mathbf{a}$	Adjoint state
argmax	Argument of the maximum
argmin	Argument of the minimum
b	Bias
C	Integration constant
$\mathbf{C}$	Global latent variable
det	Determinant
diag$(\cdot)$	Diagonal matrix with the given vector values
$\mathcal{D}_{KL}\left[q \mid\mid p\right]$	KL divergence between distributions q and p
$\mathcal{D}_{\texttt{pos}}$	Position decoder
$\mathcal{E}_{\texttt{pos}}$	Position encoder
$\mathcal{E}_{\texttt{vel}}$	Velocity encoder
$\mathbb{E}[\cdot]$	Expectation
f	Arbitrary mapping function
f^*	True mapping function
$\mathbf{f}$	Differential vector field / Latent acceleration field
$\mathbf{F}$	Bijective transformation
h	Hidden layer
h	Initial height
kg	Kilogram
l	Length of the rod
$\mathcal{L}_{\theta,\phi}$	Evidence lower bound
$\mathcal{L}_{KL}$	KL divergence
$\mathbb{L}(\cdot)$	Loss function
L	Sample size
m	Meter
$\mathbf{m}$	Amortized inference length

n	Number of balls
N	Number of data points
$\mathcal{N}(0, \mathbf{I})$	Isotropic multivariate Gaussian distribution
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and variance matrix $\boldsymbol{\Sigma}$
$\mathcal{O}$	Time complexity
$\mathbf{p}$	Parameters for Bernoulli distribution
$p(\cdot)$	Probability density function
$q(\cdot)$	Variational probability density function
s	Second
$\mathbf{s}_t$	Latent position variable at time t
$\mathcal{S}$	ODE solver
t	Time index
T	Sequence length
Tr	Trace
v	Initial velocity
$\mathbf{v}_t$	Latent velocity variable at time t
W	Weight
$\mathcal{W}$	Weight of Bayesian neural network
x	Input
$\mathbf{x}$	Multivariate random variable
y	Output
$\hat{y}$	Predicted output
$\mathbf{z}$	Latent variable
α	Initial angle
β	Regularization hyperparameter for VAE
ϵ	Random sample from isotropic Gaussian distribution
γ	Penalization hyperparameter for ODE2VAE
λ	Step size
μ	Mean

ϕ	Variational parameter set
$\sigma(\cdot)$	Activation function
σ^2	Variance
Σ	Covariance matrix
θ	Parameter set
$\odot$	Element-wise multiplication

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
ANN	Artificial Neural Network
BNN	Bayesian Neural Network
CNF	Continuous Normalizing Flows
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
ELBO	Evidence Lower Bound
GAN	Generative Adversarial Network
GD	Gradient Descent
i.i.d.	Independent and Identically Distributed
KL	Kullback–Leibler
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
Neural ODE	Neural Ordinary Differential Equation
NLL	Negative Log-likelihood
NN	Neural Network
ODE	Ordinary Differential Equation
ODE2VAE	Ordinary Differential Equation Variational Autoencoder
ODE2VAE-c	Ordinary Differential Equation Variational Autoencoder with Global Representation
PSNR	Peak Signal-to-Noise Ratio
ReLU	Rectified Linear Unit
ResNets	Deep Residual Networks
RNN	Recurrent Neural Network
SDE	Stochastic Differential Equation

SGD	Stochastic Gradient Descent
SI	International System of Units
VAE	Variational Autoencoder
VI	Variational Inference
β-ODE2VAE	Beta-Ordinary Differential Equation Variational Autoencoder
β-VAE	Beta-Variational Autoencoder

1. INTRODUCTION

1.1. Motivation and Contributions

Deep generative models such as variational autoencoders (VAEs) [1] and generative adversarial networks (GANs) [2] aim to learn distributions that represent and generate the data [3]. The learned distribution can be used for generating new data [4], extrapolating into the future [5, 6], data imputation [7], representation learning [8, 9], anomaly detection [10], clustering [11], and multi-modal learning [12]. It is interesting to learn these distributions by using unsupervised learning algorithms which do not require any labeled data. One of the challenging tasks for deep generative models is learning high-dimensional probability distributions over sequential data such as videos, speech or music.

Deep latent variable models learn a joint probability distribution over the data and the low-dimensional latent variables which represent underlying generative factors of the data. These latent variables are not only useful for the generative process but they also capture the hidden structure of the data. Although deep latent variable models are black-box architectures, the low-dimensional latent representations may be semantically meaningful and interpretable if the model is regularized and has a proper inductive bias in it [13, 14]. For instance, there are recent works on embedding physics-motivated inductive biases in deep learning models [15, 16].

In this thesis, we work on understanding and extending a continuous-time generative model, deep generative second order ordinary differential equations (ODEs) with Bayesian neural networks (ODE2VAE) [5]. The ODE2VAE model learns latent trajectories of videos by using second order latent ODEs [17]. The ODE2VAE model focuses on learning arbitrary dynamics of three sequential datasets: CMU walking data, rotating MNIST, and bouncing balls datasets.

The CMU walking dataset consists of sequences of a fixed number of joint angle measurements, the rotating MNIST dataset has the image sequences of handwritten “3” digits, and the bouncing balls dataset has videos of three balls bouncing in a two dimensional (2D) box. The baseline model focuses on developing better reconstruction and extrapolation performance. Although the model has improved metrics over the described datasets, the representations learned by the model are overlooked. Given that the model has a physics-motivated inductive bias, it has the capacity of modeling well-defined physical dynamics (e.g. bouncing balls) and learning latent representations that may capture approximate physical generating factors. This property can be explored by challenging the model with different motion datasets and checking if the latent representations behave according to the physical intuitions. Moreover, since the baseline model is tested with datasets that have fixed content, the true generative factors have a single set of time-independent factors. Therefore, the model's capacity of learning the content of the sequences is not investigated. Since the model has dynamic latent states, learning static features of a video and preserving them over time steps may be a compelling task. Two possible approaches that may help learning static features are regularizing the dynamic latent units for capturing static features or extending the baseline architecture properly.

In this thesis, we have made the following contributions:

- We investigate ODE2VAE's performance on modeling the three motion datasets: bouncing balls, projectile motion, and simple pendulum.
- We analyze the model's dynamical latent representations during different motion types.
- We uncover the effects of the inductive bias of the ODE2VAE model over its learned dynamical latent representations.
- We extend the model formulation for flexible regularization and we extend the model architecture to learn the static features efficiently.
- We investigate the performance of the baseline model on the datasets with varying static features.

We tested the baseline model's performance with three different motion datasets in order to quantify its generalizability capacity and to uncover the effects of its inductive bias over the learned representations. We show that the ODE2VAE model can learn physically plausible latent representations without any supervision. After we reformulate the baseline model, we attempt to regularize it, which has the effect of increasing the approximate disentanglement of the time-variant and invariant features. We also introduce another model variant that achieves similar disentanglement with a new architecture design. Lastly, we aim to compare the performances of the baseline model and the proposed model variants on the three novel datasets with varying static features. Due to the convergence issues, we are only able to investigate the performance of the baseline model. We show the dynamical inductive bias of the ODE2VAE model may be a disadvantageous property that hinders learning varying static features.

1.2. Organization of the Thesis

The thesis is organized as follows: In Chapter 2, fundamental deep learning notions are summarized. In Chapter 3, a comprehensive theoretical background and related works about variational autoencoders [1], disentangled representation learning [18, 19], neural ordinary differential equations [17], Bayesian Neural networks [20], and ODE2VAEs [5] are given. In Chapter 4, the proposed ODE2VAE model variants are presented. In Chapter 5, the results section is presented for elaborating on the datasets, evaluation metrics, implementation details, model hyperparameters, and experiment results. Lastly, comments and discussions about the results and possible future works are presented in Chapter 6.

2. FUNDAMENTAL BACKGROUND

The learning tasks in Machine Learning (ML) can be defined as a mapping between input space and output space . Supervised learning algorithms require using examples of data instances and labels, $\{(x_i, y_i)\}_{i=1}^{N}$, which are input and output pairs from the true mapping function f^* that is aimed to learn. Training an ML model for a supervised learning task is equivalent to learning a model f with the parameters of θ that approximates the true mapping function f^*. For instance, given images and the number of people in them as example pairs, an ML model can be optimized as a function approximation that takes images as input and outputs the number of people in them. Unsupervised learning, on the other hand, aims to learn unobserved structure of data using examples of $\{(x_i))\}_{i=1}^{N}$. The approximated model can be used for generating new data or modeling the hidden variables of the underlying data generation process. For example, learning how to compress images to a lower-dimensional representation space and decompressing them back to the data space may be used for memory-efficient storage of images [21].

The learning task can be seen as an optimization procedure with an objective function to be minimized with respect to model parameters, θ. The objective function, also called loss function L, computes a distance metric between ground truth labels y and model predictions $\hat{y} = f_\theta(x)$. Since we do not have complete access to the true data generating distribution, the loss function is computed over a limited set of observations. They are called training and test sets, which are assumed to be independent and identically distributed (i.i.d.) samples from the true data generation distribution. The loss function can be computed over the training set, $\{(x_1, y_1), \ldots (x_N, y_N)\}$, as follows:

$$f^* \approx \arg\min_{f_\theta} \frac{1}{N} \sum_{i=1}^{N} L\left(f_\theta\left(x_i\right), y_i\right). \tag{2.1}$$

It is important to highlight that it is generally not possible to learn the true mapping function itself but a proxy function, since there is a limited available data points and computation power of our model. There exist different optimization techniques that help ML practitioners find best model parameters such as computing closed form solutions (if they exists), exhaustive hyper-parameter search, and gradient based learning. We limit our discussions on gradient based learning because the model architectures that are mentioned in this thesis are suitable for gradient based learning. Therefore, the learning procedure is equivalent to finding the best model parameters that approximate the true model f^* by minimizing the error metric that is computed [22].By using the loss function, it is possible to compute gradients with respect to each model parameter by using the chain rule. Since the gradients with respect to parameters show a direction of the steepest ascent of the error metric, updating the parameters in the direction of the negative gradient proportional to the chosen step size minimizes the error metric. This iterative optimization is called gradient descent (GD) which is a first-order method. There is also a stochastic variant of GD algorithm called Stochastic Gradient Descent (SGD) [23] that is performing gradient descent given the error metric computed over a minibatch of example pairs, and repeating that by going over all examples with the assumption of the data are i.i.d. There are also advanced optimization algorithms such as Adagrad [24], and Adam [25], which allow one to change the step size during optimization. It is worth highlighting that the optimization problem cannot be solved analytically because the training data, consisting of limited set of example pairs, does not cover all possible mappings between input and target space. Therefore, the learned model, function approximation f_θ, would be a good approximation but generally not the function f^* itself, in Equation (2.1).

2.1. Deep Learning

Deep Learning (DL) is a subfield of Machine Learning which aims to build models that are capable of learning abstract representations of input data [22,26]. DL models utilize a stacked layer of computations to increase the level of abstraction in representations.

These abstract representations are learned automatically for the given task by updating the model weights with back-propagation [23]. DL models require a vast amount of data and considerable computational power. They have become promising and advantageous compared to other ML models because the amount of computation power and available data increase drastically. DL models are composed of Artificial Neural Networks, which are defined in the following section.

2.2. Artificial Neural Networks

Artificial Neural Networks (ANNs) are composed of multiple computational units that are designed with taking advantage of signal transmission in nervous systems in biology [27]. ANNs consist of layers of artificial nodes. These nodes reside in layers and have weighted connections to the nodes in the next layer. By using the node values in the given layer n and weighted connections W^n that connect them to the nodes in the following layer, the node values in the layer $n+1$ can be computed by using matrix multiplications followed by non-linear activation functions. The first and last layers of an ANN are called input and output layers, respectively. The input layer passes x through the network and the output layer outputs y as a network prediction. The layers between them are called hidden layers. In a supervised setting, ANNs can be used as function approximators, f_θ, that learn to a true mapping, f^*, between input space X and output space Y. The approximation f_θ is learned by using a set of paired examples $(x_1, y_1), ...(x_N, y_N)$ with N elements. The task of learning can be defined as determining the parameters of the neural network θ, weights, and biases. Learning procedure requires updating the network using the gradients propagation loss function with respect to network weights. This is called the back-propagation algorithm, and it empowers training neural networks efficiently with gradient-based learning [23]. From a mathematical perspective, many research studies show that neural networks can approximate a wide range of functions with a sufficient number of hidden layers, nodes, and non-linear activation functions [28, 29]. It is worth noting that sparsity of the connections or stochastic weight values imposes different neural network architectures, which will be discussed in the following sections.

2.2.1. Multilayer Perceptron

Multilayer perceptron (MLP) are fundemental neural networks that can be used for f_θ. A multilayer perceptron has fully connected layers with hidden layers, weights (W^n), bias terms (b^n), and activation functions ($\sigma(\cdot)$).

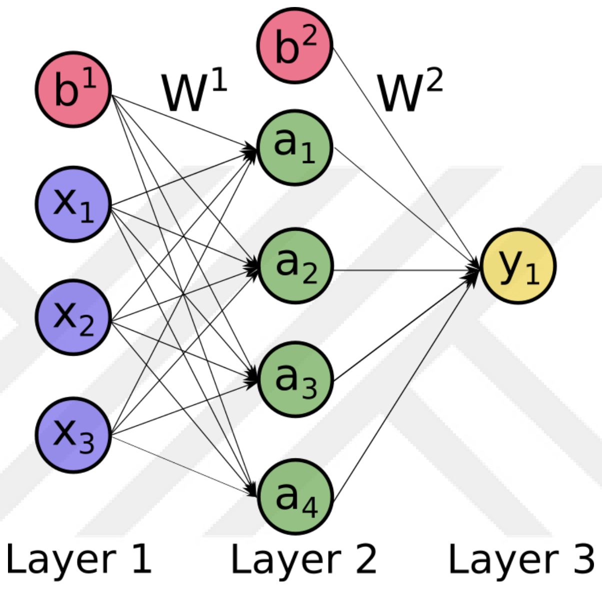


Figure 2.1. Multilayer perceptron with single hidden layer. Layers 1,2, and 3 correspond to input, hidden, and output layers, respectively. The network parameters are the weights W^n, and biases b^n.

For example, Figure 2.1 shows a single hidden layered MLP which can be implemented by using affine transformations followed by non-linear activation functions. Figure 2.1 includes three main concepts: matrix multiplications, non-linear activation functions, and bias nodes. A common practice is using vector notation for node values at each layer. Hidden node values of the next layer h_i can be computed as:

$$h_i = \sigma(W^{i^\top} h_{i-1}) \tag{2.2}$$

where bias nodes are concatenated to the hidden layers before every forward computation.

As shown in Figure 2.1, bias nodes are not connected to previous layers through weights and they are concatenated to the node values at each layer except the last one. They help to shift the output value of matrix multiplication. Since weighted connections are insufficient for modeling non-linear relationships, there are also non-linear activation functions used on top of the affine transformation that is computed by the previous layer's real-valued nodes and weights. It can be shown that using a stack of hidden layers without non-linear activation functions can be equivalently modeled with a single hidden layer. Therefore, non-linear activation functions are a crucial part of the learning task. Some of the common activation functions that are used in hidden layers are rectified linear unit (ReLU) $max(0, x)$, sigmoid function $1/\left(1 + e^{-x}\right)$, tanh, and softmax function $e^{x_i}/\sum_{j=1}^{K} e^{x_j}$ which is generally used in the output layer.

2.2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) also have hidden layers, weights, biases, and activation functions as MLPs have. However, they are specifically designed to work with spatial inputs such as images, videos, or volumetric data. Its hidden or output layer may also be three-dimensional. CNNs do not have fully connected layers; instead, they have a filter of weights that convolves a given window, sliding over the inputs with a given stride. The depth of the layers can be thought of as stacked outputs of filters in each layer. Since CNNs convolve input with filters, it is equivalent to using the same set of weights in various parts of the input. The common intuition behind this property is having multiple filters that look for local features over the input, which establishes the spatial invariance property of CNNs. Therefore, CNNs can extract features by using less number of learnable parameters compared to MLPs. Stride of the layer defines the step size of the convolution operation. Padding refers to filling the border of the input tensor with zeros, which is generally used to control the shape of the output map. Another important operation used with CNNs is pooling, which corresponds to convolving the input with a filter that downsizes the receptive field by applying the max operation [22]. It is worth mentioning that CNNs may perform three dimensional (3D) convolution for extracting features from temporal data.

CNNs are generally preferred for extracting local features. In Figure 2.2, the input image with size $3 \times 32 \times 32$ is convolved with stride 1 by a filter with size $3 \times 5 \times 5$ that stores network weights. The result, activation map, is computed by the matrix multiplication of the filter and the receptive field over the image. The output has the size $1 \times 28 \times 28$. Therefore, each pixel corresponds to different dot products during the convolution operation.

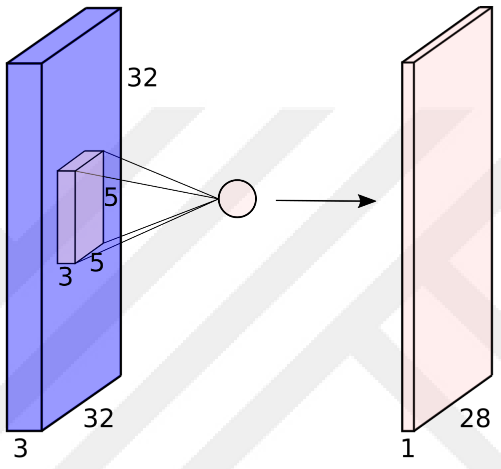


Figure 2.2. Illustration for convolution operation. The input image is convolved by a filter to produce the output image.

CNNs are utilized for various tasks that primarily deal with visual information. One of the earliest applications of CNNs that are trained using back-propagation is for image recognition, recognizing handwritten numbers [30]. One of the well-known applications of CNNs is AlexNet [31], which was the first neural network state-of-the-art model on the ImageNet challenge [32] for object detection and image classification. It has showed how successful NNs can be used to solve recent challenges in ML world.

3. THEORETICAL BACKGROUND

In this thesis, we mainly deal with generative models, which are trained in an unsupervised way to learn complex probability distribution $p^*(\mathbf{x})$, which is the underlying probabilistic process for generating high-dimensional data $\mathbf{x}$. In this setting, $p^*(\mathbf{x})$ is unknown and there are limited independently and identically distributed (i.i.d.) observations $\{(x_i)\}_{i=1}^N$ that are sampled from the true distribution. The task of learning is approximating the true distribution $p^*(\mathbf{x})$ by finding the best model $p_\theta(\mathbf{x})$ with the set of parameters θ that approximate the underlying process as follows:

$$p^*(\mathbf{x}) \approx p_\theta(\mathbf{x}) \tag{3.1}$$

$$\mathbf{x} \sim p_\theta(\mathbf{x}) \tag{3.2}$$

$$\{(\mathbf{x_i})\}_{i=1}^N \overset{\text{i.i.d.}}{\sim} p^*(\mathbf{x}) \tag{3.3}$$

where $\sim$ denotes the distribution of the random variables. This is a challenging task because it is needed to design $p_\theta(\mathbf{x})$ flexible enough to be able to approximate the true distribution. After the best set of model parameters θ are found, the model can use them to generate new samples $\mathbf{x}$. For instance, a real-life application may be learning the distribution over the X-ray scans of people. In this example, $p_\theta(\mathbf{x})$ may be the candidate model to learn correlations between pixel values of scans. Since the inputs are high-dimensional scans, the true distribution is a complex distribution to learn. It may be beneficial to utilize latent variable models. Latent variables, $\mathbf{z}$, are hidden variables that are not observed during the data generation process. By using a latent variable model, we can still recover $p_\theta(\mathbf{x})$ by marginalizing out latent variables, which can be formulated as:

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x} \mid \mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z}) \tag{3.4}$$

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}. \tag{3.5}$$

In a latent variable model, the generative model is equivalent to learning the parameters of joint distribution of observations and hidden variables, $p_\theta(\mathbf{x}, \mathbf{z})$. The term $p_\theta(\mathbf{x})$, is referred as marginal likelihood of data or evidence. Since the complex nature of $p_\theta(\mathbf{x})$ can be hard to model, latent variable models enable us to factorize the marginal likelihood as the product of prior and likelihood distributions (see Equation (3.4)). By introducing a factorized form of the joint distribution, the marginal likelihood can be constructed as the product of simpler distributions which enables us to approximate the true distribution $p^*(\mathbf{x})$ through a simpler and more tractable latent variable model (see Equation (3.5)). The latent variable models used in this thesis are deep latent variable models that are parameterized by neural networks. By using Bayes' theorem, it is possible to infer latent variables of a given data point:

$$p_\theta(\mathbf{z} \mid \mathbf{x}) = \frac{p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}. \tag{3.6}$$

Although the marginal likelihood (or evidence) can be factorized, it requires high dimensional integrals to compute (see Equation (3.5)), which makes the denominator and the posterior computation intractable in Equation (3.6). In the next section, variational inference methods are introduced. They are utilized to overcome the intractability issue. By overcoming the intractability problem, it becomes possible to infer latent representations of a given data point through the posterior. The inferred latent representations can be thought of as the low dimensional and abstract hidden variables that are needed to generate the data itself. The latent representations may be used for a wide range of tasks such as representation learning [8], clustering [11], sequence modeling [33], image classification [34], and multi-modal learning [12].

3.1. Variational Inference and Variational Autoencoders

Variational Autoencoders (VAE) [1] are deep latent variable models which approximate the generative distribution $p_\theta(\mathbf{x})$ through latent variables. The main motivation behind VAEs is using variational inference (VI) [35] to model $p_\theta(\mathbf{x})$.

VI aims to find approximate posterior which cannot be solved analytically due to the intractability in Bayes' theorem imposed by the marginal likelihood in the denominator (see Equation (3.6)). Variational inference methods reformulate the approximate posterior problem as an optimization problem [36] as shown in Figure 3.1. In Figure 3.1, $q(z \mid x; \phi)$ denotes the set of approximate variational distributions where ϕ denotes the parameters.

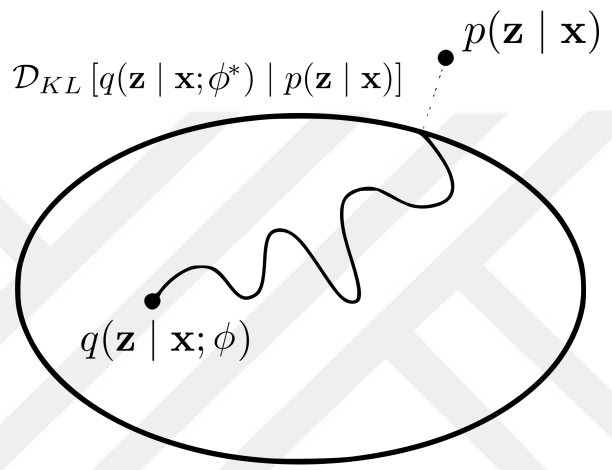

$p(\mathbf{z} \mid \mathbf{x})$
$\mathcal{D}_{KL}\left[q(\mathbf{z} \mid \mathbf{x}; \phi^*) \mid p(\mathbf{z} \mid \mathbf{x})\right]$
$q(\mathbf{z} \mid \mathbf{x}; \phi)$

Figure 3.1. Illustration for variational inference methods. The aim is to find the best parameters ϕ^* that minimize the KL distance between the variational approximation (q) and the target distribution (p). The figure is adapted from [37] with permission.

The optimization scheme starts with defining the set of possible solutions to search for the best approximation. To be more specific, a family of distributions is selected to define the set of possible approximate distributions. The aim of the optimization procedure is finding the best distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$, with variational parameters ϕ [1], that approximate the posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$ as follows:

$$q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \approx p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x}) \tag{3.7}$$

which aims to find variational parameters ϕ ensuring the tractability of the posterior in Equation (3.6).

The objective function of this optimization problem is Kullback-Leibler (KL) divergence, which measures the difference of a candidate and the reference distribution. The intuition behind KL divergence is calculating a metric that shows how much information is lost when the candidate distribution is used instead of the reference distribution. KL divergence between variational posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ and intractable posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$ is minimized w.r.t. variational parameters:

$$q^*_\phi(\mathbf{z} \mid \mathbf{x}) = \arg\min_\phi \mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z} \mid \mathbf{x})\right]. \tag{3.8}$$

KL divergence can be written as:

$$\begin{aligned}\mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z} \mid \mathbf{x})\right] &= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \left[\log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z} \mid \mathbf{x})}\right] d\mathbf{z} \\ &= \mathbb{E}_{q_\phi}\left[\log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z} \mid \mathbf{x})}\right].\end{aligned} \tag{3.9}$$

One can check that the KL divergence of two distributions is always non-negative [38]. It is equal to zero if two distributions are identical, which is equivalent to finding a variational posterior that can replace the true posterior. With this optimization procedure in mind, the relationship between KL divergence and marginal log-likelihood can be formulated as the following:

$$\begin{aligned}\mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z} \mid \mathbf{x})\right] &= -\mathbb{E}_{q_\phi}\left[\log \frac{p_\theta(\mathbf{z} \mid \mathbf{x})}{q_\phi(\mathbf{z} \mid \mathbf{x})}\right] \\ &= -\mathbb{E}_{q_\phi}\left[\log \frac{p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} - \log q_\phi(\mathbf{z} \mid \mathbf{x})\right] \\ &= -\mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})\right] \\ &\quad + \log p_\theta(\mathbf{x}).\end{aligned} \tag{3.10}$$

In Equation (3.10), logarithm of the evidence can be pulled out of the expectation operation since it is does not depend on the variational posterior. Therefore, it can be organized as follows:

$$\begin{aligned}\log p_\theta(\mathbf{x}) &= \mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z} \mid \mathbf{x})\right] \\ &\quad + \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})\right] \\ &= \mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z} \mid \mathbf{x})\right] \\ &\quad + \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathbb{E}_{q_\phi}\left[\log \frac{\log q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z})}\right]. \end{aligned} \tag{3.11}$$

The left-hand side of Equation (3.11) has data log-likelihood, which does not depend on hidden variables and should be maximized for approximating the true distribution in (Equation (3.1)). The right-hand side of Equation (3.11) consists of the KL divergence term, which is non-negative and needs to be minimized during variational inference. Therefore, all the other terms on the right-hand side can be seen as a lower bound for the data log-likelihood, which is called the evidence lower bound (ELBO):

$$\begin{aligned}\log p_\theta(\mathbf{x}) &= \mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z} \mid \mathbf{x})\right] \\ &\quad + \underbrace{\mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathbb{E}_{q_\phi}\left[\log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z})}\right]}_{\text{(ELBO)}} \\ &= \mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z} \mid \mathbf{x})\right] + \underbrace{\mathcal{L}_{\theta,\phi}(\mathbf{x})}_{\text{(ELBO)}} \end{aligned} \tag{3.12}$$

and the ELBO term can be written as:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathbb{E}_{q_\phi}\left[\log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z})}\right]. \tag{3.13}$$

Maximizing the ELBO term has two crucial effects on the optimization [39]. Firstly, it increases the data log-likelihood. Increasing data log-likelihood $\log p_\theta(\mathbf{x})$ is equivalent to increasing marginal likelihood since logarithm is a monotonic function.

Secondly, it forces the KL divergence between the variational approximate posterior and the true posterior to be smaller, which generates a better variational approximate posterior.

VAEs can be used to tackle this variational inference problem by using two neural networks and a gradient-based learning algorithm to optimize both neural networks jointly (see Figure 3.2). They are trained by maximizing the ELBO in Equation (3.13) or, equivalently, minimizing negative ELBO.

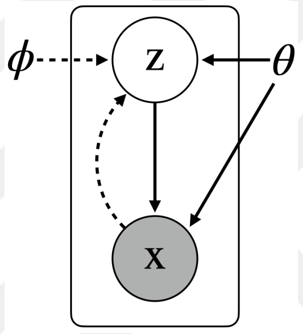


Figure 3.2. The graphical model representation of variational autoencoder model. θ and ϕ represents the set of parameters for generative and approximate variational distribution, respectively.

The first neural network is named as the encoder (inference network), which parameterizes the variational posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$. In this formulation, ϕ corresponds to the weights and biases of a neural network. It outputs a distribution over latent variables $\mathbf{z}$, given the input $\mathbf{x}$. The output distribution is often chosen as a multivariate Gaussian distribution with parameters $\boldsymbol{\mu}, \boldsymbol{\sigma}$ and a diagonal covariance matrix:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \text{Encoder}_\phi(\mathbf{x}) \tag{3.14}$$

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})) \tag{3.15}$$

$$\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x}). \tag{3.16}$$

Multivariate Gaussian distribution with random variable $\mathbf{x} \in \mathbb{R}^k$, mean $\boldsymbol{\mu} \in \mathbb{R}^k$, and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{k \times k}$ is given as:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right). \tag{3.17}$$

In contrast to traditional VI, the parameter set of the encoder model, ϕ, is shared among all data points $\mathbf{x}$, which is called amortization of weights [40]. Amortized VI has advantages over traditional VI, such as using fewer parameters and directly computing variational parameters of new data without running optimization steps. Therefore, it is scalable and efficient but also has a limited capacity for approximating the posterior.

The second neural network is a decoder, which is the generative model that models the joint distribution of data and latent variables, $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z})$. The prior distribution $p_\theta(\mathbf{z})$ is generally chosen as a multivariate Gaussian with mean of zero and covariance matrix of identity (see Equation (3.18)). The conditional distribution or likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$ is parameterized by a neural network with parameters θ (see Equation (3.20)). It models a distribution over data points, which may be used for reconstruction given a latent variable $\mathbf{z}$. The family of the likelihood distribution is chosen according to the properties of the data. For instance, when dealing with data consist of binary images, Bernoulli distribution over each pixel value is used as the likelihood distribution [1]. As mentioned before, the main motivation behind the factorization of the joint distribution is modeling the complex marginal likelihood by using the product of two tractable distributions, which are:

$$p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I}) \tag{3.18}$$

$$\mathbf{p} = \mathrm{Decoder}_\theta(\mathbf{z}) \tag{3.19}$$

$$p_\theta(\mathbf{x} \mid \mathbf{z}) = \mathcal{B}(\mathbf{x}; \mathbf{p}). \tag{3.20}$$

Multivariate Bernoulli distribution is denoted with $\mathcal{B}(\mathbf{x};\mathbf{p})$, which is given as:

$$\mathcal{B}(\mathbf{x};\mathbf{p}) = \prod_i \mathcal{B}\left(x_i;p_i\right) = p_i^{x_i}(1-p_i)^{1-x_i}. \tag{3.21}$$

There are two neural networks in a VAE that should be trained jointly by maximizing the objective function, ELBO. The aim is to train the model efficiently with gradient-based learning by using minibatch SGD [23, 41]. However, the architecture of VAEs brings a major challenge regarding the back propagation and parameter optimization steps. The first neural network, encoder, is a conditional distribution over latent variables. It means that, in a forward computation, the latent variable that is used in the decoder is sampled from the conditional distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$. During the back-propagation, it is needed to compute the derivative of the random variable $\mathbf{z}$ w.r.t. encoder parameters ϕ for stochastic back-propagation [1, 42]. Since it is not possible to take derivatives of the random variable, the sampled latent variable $\mathbf{z}$ is reparameterized, which is called the reparameterization trick [1, 42]. Figure 3.3 illustrates the reparameterization trick.

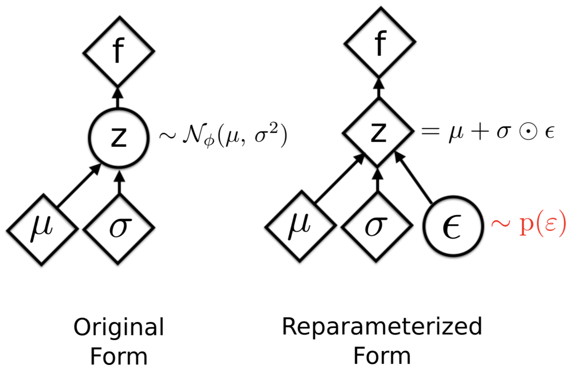


Figure 3.3. The graphical model describing reparameterization trick. Deterministic and stochastic nodes are denoted with squares and circles, respectively. The figure is adapted from [43] with permission.

The main idea of the reparameterization trick is enforcing the randomness of the latent variable $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x})$ to be originated from a separate node of noise, which makes $\mathbf{z}$ a deterministic node with random noise so that it is possible to take derivatives w.r.t. encoder parameters ϕ. For instance, for a given mean and standard deviation $(\boldsymbol{\mu}, \boldsymbol{\sigma})$, latent variable $\mathbf{z}$ can be re-written as:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{3.22}$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{3.23}$$

where $\odot$ denotes element-wise multiplication. The definition in Equation (3.23) allows one to back-propagate. Furthermore, during optimization, the ELBO term in Equation (3.13) can be computed by using log probabilities of the distributions in Equations (3.15), (3.18), and (3.20). It is possible to decompose the ELBO itself to give some intuition about what VAEs try to optimize. The ELBO term can be written as two main terms; a reconstruction term and a regularization term:

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathbb{E}_{q_\phi}\left[\log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z})}\right] \\ &= \underbrace{\mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right]}_{\text{Reconstruction}} - \underbrace{\mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z})\right]}_{\text{Regularization}}.\end{aligned} \tag{3.24}$$

The reconstruction part drives the model to learn meaningful latent representations that can be decoded to output parameters of the conditional likelihood distribution. The conditional distribution can then be used for data generation. The KL term shows that the variational posterior is regularized by the prior distribution, which is generally selected as the standard normal distribution. It forces the learned variational posterior to overlap with the prior, which builds a smooth latent space. This setting encourages creating a latent space where encodings of similar data points are located nearby. The intuition behind these latent clusters stems from the probabilistic nature of the model. Since the latent representations are sampled from a distribution, there is a possibility of sampling various latent representations given an arbitrary data point x.

The model is optimized to maximize the log-likelihood. Therefore, the model loses the least amount of the reconstruction likelihood if the sampled z is decoded to generate parameters for a similar data point x [44].

VAEs are utilized for various research problems such as learning generative models for image datasets, i.e., handwritten digits, faces gestures, house numbers [1, 45]. They are also used for image classification [34], object segmentation [46], clustering [9, 11], anomaly detection [10], future forecasting [47, 48],sequence modeling [33], time series imputation [7], representation learning [8, 9], disentangled representations [49], zero and few-shot learning [50], and multi-modal learning [12].

3.2. Disentangled Representation Learning and β-VAE

The field of representation learning deals with learning representations that are useful for down-stream tasks such as classification, forecasting, prediction, and compression [51]. In a probabilistic setting, learning better representations enables probabilistic models to learn better posterior distributions over the hidden generating factors of the data [51]. In this thesis, we are interested in learning representations with deep learning models in a probabilistic setting. Therefore, we generally refer to latent variables as representations that are learned. Since representations carry necessary information about the factors that generate high-dimensional data, they have typically lower dimensions compared to data points. For instance, for 32×32 images of a bouncing ball that moves in two-dimensional space, there are three generating factors, x and y coordinates, and radius of the ball. In this example, learning useful representations corresponds to extracting low-dimensional representations that are related to true hidden factors of the data, such as position coordinates and the radius. It is worth highlighting that learning representations does not aim to recover true hidden variables themselves but their useful transformations.

Disentangled representation learning carries the aim a step further and aims to learn representations that are disentangled, so they are coupled with a single generative factor and invariant to the others [19,51] i.e., disentangling object shape, color, position, and background color. It brings the advantage of learning latent representations that are semantically meaningful and interpretable, which correspond to the transformation of true generative factors. These properties are very useful for down-stream tasks and building interpretable models [52]. As already mentioned, VAEs are generative models that aim to learn a joint distribution over data points and latent representations [1]. Since latent encodings in a VAE are used for decoding the data itself, they are indeed learned generative factors in an unsupervised setting.

β-VAE is a VAE variant that deals with visual data and its disentangled latent representations by modeling a disentangled posterior [18,44]. β-VAE requires unsupervised training like as a traditional VAE. It uses the original ELBO term for VAEs (see Equation (3.24)) and parameterizes the KL term in it with a hyperparameter β:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \beta\, \mathcal{D}_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z})\right]. \tag{3.25}$$

When $\beta > 1$, the KL term in Equation (3.25) contributes more, which enforces the regularization of the approximate posterior by creating an information bottleneck given the prior distribution as multivariate standard normal [18]. Since the prior distribution has independent components among its dimensions, the KL divergence forces elements of the approximate posterior to be factorized, too. The most useful latent units, which are crucial for reconstruction and maximizing the ELBO objective, create a non-zero KL divergence with the unit Gaussian prior. The latent units, which do not contribute to the reconstruction objective, do not fit in the information bottleneck. Therefore, they minimize the KL term and stay closer to the standard Gaussian prior independently. Increasing the β hyperparameter forces disentangled representations, but it may cause increased errors among reconstructions after an arbitrary value [18]. For example, when working with image data, it can generate blurry data reconstructions.

When the β coefficient gets too large, it does not allow storing any information in latent space. The optimal value for β requires fine-tuning the trade-off between reconstruction ability and disentangling latent representations. This value changes according to different datasets and model architectures [18]. It is also observed that when the β is large, there is a strong constraint on the latent bottleneck; the latent spaces only capture the most prominent generative factors [44]. The reason is the model tries to learn these prominent factors in the latent space, which create the highest possible reconstruction likelihood under the constrained latent bottleneck [44]. Therefore, it is possible to start with a high β value and decrease it during training. It helps to learn factorized latent factors by starting from the most important to the least important one.

Although we use the β-VAE formulation in this thesis, there are other β-VAE variants for disentangled representation learning such as FactorVAE [53] and β-TCVAE [54], which decompose the KL term and regularize specific subcomponents. Also, InfoGAN [55], a generative adversarial network (GAN) variant [2], disentangles latent representations by maximizing the mutual information between the latent representations and their outputs.

Disentangled representations in low-dimensional latent spaces may be informative about how dynamics of the process evolves. In deep learning, there is a high number of possible model designs for solving a specific task, and some of these model choices perform better than others [13]. This can be explained by the inductive bias of the models. Inductive bias corresponds to a set of assumptions that are imposed to a model before any training. Inductive bias may increase the interpretability and generalization of the model [13,14]. For instance, CNNs have an inductive bias for being invariant to spatial translation. The latent representations may be more informative when disentangled representations in low-dimensional latent spaces are regularized with proper inductive bias. For instance, videos of a dynamical system can be modeled in a low-dimensional manifold [56–58]. It is possible to disentangle static and dynamics components of a video and generate new videos by swapping components of a video [59].

For a video of a physical system, this may help modeling representations of physical quantities such as position and velocity given that the model has appropriate inductive biases, i.e., position latents are driven by the velocity latents according to equations of motion. In this example, the inductive bias embeds intuitive physics concepts to the model [5]. It can also be used for embedding more complicated physical concepts [15, 16].

3.3. Neural Ordinary Differential Equations

Thanks to advances in variational inference and deep learning, VAEs have been used for extracting meaningful data representations and understanding the latent space and the data manifold [1]. There are also recurrent neural network (RNN) based encoders and decoders that can be used to extract representations of sequential data [60, 61]. The main problem of the mentioned models is that the inputs are discrete and regularly sampled, whereas real-world datasets or observations are continuous by nature. Building deep learning models that can model continuous functions [6, 62, 63] has the benefits of working with irregularly sampled data.

A recent work, Neural Ordinary Differential Equations (Neural ODEs) [17] attempt to tackle this issue of continuity. Rather than operating in discrete-time, it enables deep learning models to operate on continuous data. Ordinary differential equations are equations that relate the functions of a single independent variable $z(t)$ and its derivatives $\{\dot{z}(t), \ddot{z}(t), \dots\}$ with respect to the same single independent variable t. In this thesis, we generally deal with first-order differential equations in which only the first-order derivative is present. The aim is solving for the function of the independent variable $z(t)$ by using the relationship imposed by the ODE. The function of the independent variable may also be called the dependent variable $z(t)$. For a dependent variable $\mathbf{z} \in \mathbb{R}^D$, the differential function or vector field $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$ creates a gradient field in data space of the dependent variable.

This vector field denotes the continuous dynamics of a system e.g., how dependent variable $\mathbf{z}(t)$, its derivative $\dot{\mathbf{z}}(t)$, and independent variable t change simultaneously:

$$\dot{\mathbf{z}}(t) = \frac{d\mathbf{z}(t)}{dt} = \mathbf{f}(\mathbf{z}(t), t) \in \mathbb{R}^D. \tag{3.26}$$

By integrating the differential function, the dependent variable can be computed as a function of the independent variable, which is called the general solution for the ODE that is in consideration:

$$\mathbf{z}(t) = \int \mathbf{f}(\mathbf{z}(\tau), \tau) d\tau + C. \tag{3.27}$$

The general solution is valid for any value of constant C. Therefore, it does not offer an exact solution until C is determined. Determining the value of C is possible when a boundary condition or initial value of the dependent variable is provided on top of the differential field in Equation (3.26):

$$\mathbf{z}(0) = \mathbf{z}_0 \tag{3.28}$$

$$\mathbf{z}(t) = \mathbf{z}_0 + \int_0^t \mathbf{f}(\mathbf{z}(\tau)) d\tau. \tag{3.29}$$

In order to compute the particular solution, the differential field should satisfy existence and uniqueness conditions. The existence condition checks if the differential field is continuous on the interval around the initial value. The uniqueness theorem checks if the partial derivative of the differential field is continuous on the interval around the initial value. One can check if the ODE satisfies the Lipschitz condition [64] to verify that it is well-posed, meaning that the solution exists, is unique, and stable.

In some cases, analytical solutions for ODEs may not be available. Therefore, it may be needed to utilize numerical approximations to compute the integrals in Equation (3.29). Let us assume that the ODE has time t as the independent variable and position z as the dependent variable. Numerical approximations aim to solve for $z(\tau)$ where $\tau = \{t_1, t_2, \ldots, t_n\}$ with the given initial value at time t_0.

There are plenty of numerical integrators for the numerical computation of integrals. For introductory purposes, Euler's Method, a first-order numerical approximation, is described to provide sufficient motivation. The idea of Euler's Method is approximating the solution using the local tangent values at each time step, where λ denotes the step size between adjacent time steps [65]:

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t) \approx \frac{\mathbf{z}(t+\lambda) - \mathbf{z}(t)}{\lambda} \tag{3.30}$$

$$\mathbf{z}(t+\lambda) = \mathbf{z}(t) + \lambda f\left(\mathbf{z}(t), t\right). \tag{3.31}$$

This formulation has significant resemblances with deep residual networks (ResNets) [66]. ResNets have forward propagation of the input through network layers, where each layer output is summed with its input:

$$\mathbf{y} = \mathbf{x} + h(\mathbf{x}) \tag{3.32}$$

where $\mathbf{x}$, h, and $\mathbf{y}$ denote input, hidden layer, and output respectively. These are called residual connections between input and output layers. ResNets aim to ease the training of the deep neural networks by using residual connections. It is shown that these connections solve the issues of vanishing and exploding gradients in deep neural networks. Hence, they are helpful for optimizing networks with a high number of layers [66]. Additionally, the forward propagation with residual connections in Equation (3.32) has the same functional form with Equation (3.31) as the following:

$$\underbrace{\mathbf{z}(t+\lambda)}_{\text{Output}} = \underbrace{\mathbf{z}(t)}_{\text{Input}} + \underbrace{\lambda f\left(\mathbf{z}(t), t\right)}_{\text{Residual}}. \tag{3.33}$$

This interpretation shows that deep neural networks with residual connections can be used for numerical approximations of ODEs [67]. As the step-size h gets smaller, the procedure becomes computing numerical integration continuously. It is essential to highlight that there are solvers with adaptive step-size in which step-size is tuned according to desired accuracy or tolerated error of the solution.

The discussions about ODEs, numerical approximations, and their connections with deep neural networks bring us to building deep learning models that operate continuously in time such as Neural ODEs [17]. In order to define a learning task that can operate continuously in time, the traditional learning objective $\hat{\mathbf{z}} = f_\theta(t)$ is replaced with $\widehat{\frac{d\mathbf{z}(t)}{dt}} = f(\mathbf{z}(t), t, \theta)$, where θ represents the parameters of the neural network. The aim of the neural network $f(\mathbf{z}(t), t, \theta)$ is to approximate the true dynamics of $f(\mathbf{z}(t), t)$. In other words, the aim is to learn the derivative of the function instead of the function itself. As it is necessary to write down a loss function to optimize the network with gradient-based learning, the loss function for Neural ODEs can be written as [17]:

$$\mathbb{L}[\mathbf{z}(T)] = \mathbb{L}\left[\mathbf{z}\left(t_0\right) + \int_{t_0}^{T} f(\mathbf{z}(\tau), \tau, \theta) d\tau\right] = \mathbb{L}\left[\mathcal{S}\left(\mathbf{z}\left(t_0\right), t_0, T, \theta\right)\right] \tag{3.34}$$

where $\mathcal{S}$ denotes an ODE solver.

In Neural ODEs, the ODE solver is a black-box neural network that models the dynamics by approximating the differential field [17]. In Equation (3.34), the loss function $\mathbb{L}$ computes a scalar loss metric given the initial value $\mathbf{z}(t_0)$, network parameters θ, and the interval up to time point T. For the optimization of loss $\mathbb{L}$, the network requires updating network parameters θ and $\mathbf{z}(t_0)$ by using gradients of $\mathbb{L}$ w.r.t. parameters. At this point, it is important to highlight that the initial value $\mathbf{z}(t_0)$ may also be a parameter since it may also be an output of another network such as an encoder. In this continuous setting, efficient computation of forward and backward passes is crucial. The solver that is used in the model has an adaptive step-size which implies that the number of forward and backward computations are tuned according to the problem itself. This may create memory overhead because back-propagation requires saving all middle layer outputs and computing gradients. In Neural ODEs, the memory inefficiency problem is solved by utilizing the adjoint method [68] for back-propagation. First adjoint state $\mathbf{a}(t)$ is for computing gradient of loss w.r.t. $\mathbf{z}(t)$ at each instant.

The dynamics of the adjoint state can be computed by solving another differential equation:

$$\mathbf{a}(t) = \partial\mathbb{L}/\partial\mathbf{z}(t) \tag{3.35}$$

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\top}\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} \tag{3.36}$$

$$\mathbf{a}(t_0) = \mathbf{a}\,(T) - \int_{T}^{t_0} \mathbf{a}(t)^{T}\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)}. \tag{3.37}$$

It can be seen that the adjoint state also requires network states $\mathbf{z}(t)$ over time. The ODE can be solved backwards in time in order to avoid storing states over time. Furthermore, the gradients w.r.t. network parameters θ can be computed with a second adjoint state, and it can also be computed by solving another ODE backwards in time:

$$\mathbf{a}_{\theta}(t) = \partial\mathbb{L}/\partial\theta \tag{3.38}$$

$$\frac{d\mathbf{a}_{\theta}(t)}{dt} = -\mathbf{a}(t)^{\top}\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} \tag{3.39}$$

$$\frac{d\mathbb{L}}{d\theta} = -\int_{T}^{t_0} \mathbf{a}(t)^{\top}\frac{\partial f(\mathbf{z}(\tau), \tau, \theta)}{\partial \theta}d\tau. \tag{3.40}$$

There is also another adjoint state $\mathbf{a}_t(t) = \frac{d\mathbb{L}}{dt(t)}$, which denotes gradients of loss $\mathbb{L}$ w.r.t. independent variable t, time. In [17], the authors provide an efficient algorithm to compute all of these reverse ODEs, or gradients for the whole model, in a single call to the solver by concatenating z, adjoint states, and gradients for network parameters [17]. Therefore, the memory inefficiency is solved by trading off the computation time, and by solving another auxiliary ODE backwards in time.

Up to this point, we have discussed Neural ODEs' contributions for approximating solutions of ODEs with neural networks. Their other contribution lies in the field of normalizing flows (NF) [69]. In this setting, the aim is to deal with bijective transformations of random variables through a function F and its effects on underlying distributions. Learning the transformation of the densities enables us to model complex densities by transforming simple densities such as multivariate Gaussian distribution. Thus allowing us to perform exact log-likelihood evaluation.

Also, a learned distribution can be used as an approximate posterior for finding variational approximation of the true posterior [69]. Let us assume that there is a continuous random variable $\mathbf{x}_0 \in \mathbb{R}^D$ and an underlying process $\mathbf{x}_0 \sim p_0(\mathbf{x}_0)$. There is also a bijective function $\mathbf{F} : \mathbb{R}^D \to \mathbb{R}^D$, such that $\mathbf{x}_1 = \mathbf{F}(\mathbf{x}_0)$. Change of variables formula is utilized [38, 69] to compute how the distribution $p(\mathbf{x}_0)$ changes under this transformation $\mathbf{F}$:

$$\log p_1(\mathbf{x}_1) = \log p_0(\mathbf{x}_0) - \log \left| \det \frac{\partial \mathbf{F}}{\partial \mathbf{x}_0} \right| . \tag{3.41}$$

Normalizing flows method extends the change of variable theorem in Equation (3.41). It describes the changes in the source density when the random variable from that density goes under a sequence of bijective transformations [69]:

$$\mathbf{x}_K = \mathbf{F}_N \circ \ldots \circ \mathbf{F}_2 \circ \mathbf{F}_1(\mathbf{x}_0) \tag{3.42}$$

$$\ln p_N(\mathbf{x}_N) = \ln p_0(\mathbf{x}_0) - \sum_{i=1}^{N} \ln \left| \det \frac{\partial \mathbf{F}_i}{\partial \mathbf{x}_{i-1}} \right| . \tag{3.43}$$

In Equation (3.41), the argument of the natural logarithm denotes the amount of change in a unit volume of source density under the transformation. However, computing the determinant term has a time complexity that scales in proportion to the cube of the dimension of the random variable D, $\mathcal{O}(D^3)$. Using Neural ODEs, it is possible to re-write the sequence of transformations continuously by using ODEs. In our original notation, let $\mathbf{z}(t) \in \mathbb{R}^D$ with a time-dependent probability distribution $\mathbf{z}(t) \sim p(\mathbf{z}(t))$. As in Equation (3.26), random variable $\mathbf{z}(t)$ is driven by a differential field $\frac{d\mathbf{z}(t)}{dt} = \mathbf{f}(\mathbf{z}(t), t)$. Similar to normalizing flows, the differential field describes a continuous transformation in the space of a random variable. Given the fact that the differential field is Lipschitz continuous, the differential field that drives the source density among the continuous transformation can be formalized using the instantaneous change of variables theorem [17].

The differential field and the transformation of the source density can be written as:

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\operatorname{Tr}\left(\frac{d\mathbf{f}}{d\mathbf{z}(t)}\right) \tag{3.44}$$

$$\log p\left(\mathbf{z}\left(T\right)\right) = \log p\left(\mathbf{z}\left(t_0\right)\right) - \int_{t_0}^{T} \operatorname{Tr}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}(\tau)}\right) d\tau \tag{3.45}$$

which describes how base density flows continuously through transformations. The process is called continuous normalizing flows (CNF) [17] and is trained by maximizing Equation (3.45) [70]. In Equation (3.44), there is a trace operation instead of a determinant operation, which means that the amount of change in the source distribution can be computed with time complexity scaling proportional to the square of the size of the random variable, $\mathcal{O}(D^2)$. The dynamics for random variable and its density can be solved by modeling the combined state:

$$\begin{bmatrix} \mathbf{z}\left(T\right) \\ \log p\left(\mathbf{z}\left(T\right)\right) \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{z}\left(t_0\right) \\ \log p\left(\mathbf{z}\left(t_0\right)\right) \end{bmatrix}}_{\text{initial values}} + \underbrace{\int_{T}^{t_0} \begin{bmatrix} f(\mathbf{z}(\tau), \tau; \theta) \\ \text{-Tr}\left(\frac{\partial f}{\partial \mathbf{z}(\tau)}\right) \end{bmatrix} d\tau}_{\text{dynamics}} . \tag{3.46}$$

Free-form Continuous Dynamics for Scalable Reversible Generative Models [70] is another work from the authors of Neural ODEs. It extends the approach for a more efficient sampling for computing the log-likelihood at instant T. It also reduces the time complexity of computing the log-likelihood to a level that scales linearly with the input dimension D, $\mathcal{O}(D)$ [70]. However, in this thesis, we stick to the algorithm with the time complexity $\mathcal{O}(D^2)$ for computation of log densities for CNFs.

In the recent works, Neural ODEs are analyzed to demystify the black box model [71,72]. Regularization of Neural ODEs draws attention in order to have a more stable and robust training regime [72, 73]. There are variants of Neural ODEs for different types of tasks.

Neural ODEs can be used to model the dynamics of irregularly sampled latent variables and overperforms RNN based models on irregularly sampled data [6]. They can be used for building continuous latent representations, generating continuous sequential data, and producing plausible long-term forecasts [5,6,74–76]. Neural ODEs are also utilized to learn continuous dynamics in physical systems by imposing the inductive bias that is necessary to learn physical dynamics [15, 77]. Since Neural ODEs cannot capture non-linear effects, it is possible to use two coupled Neural ODEs in order to capture non-linear system dynamics in latent dynamics [78]. Stochastic differential equations (SDE) can be interpreted as ODEs with instantaneous noise [79]. Similar to Neural ODEs, SDEs can be used for learning non-deterministic continuous dynamics [80]. It is also possible to learn weights of Neural ODEs in a Bayesian setting (see Section 3.4 for an introduction) to quantify the robustness of the model and capturing the model uncertainty [81].

3.4. Bayesian Neural Networks

Deterministic deep learning models provide point estimates for their predictions, which means that they do not capture uncertainty. They learn deterministic model parameters, which make them prone to overfitting and lacking generalization capability [20]. Moreover, capturing uncertainty and providing a confidence interval over the predictions has become significantly important [82] as deep learning models are deployed for healthcare and diagnostics [83,84], robotics [85,86], self-driving vehicles [87], and recommendation systems [88,89]. Hence, it is essential to capture the model's uncertainty about its outputs. Bayesian neural networks (BNN) present a probabilistic setting to train DNNs. For instance, BNNs are utilized in computer vision tasks to capture aleatoric and epistemic uncertainties [90]. This particular approach has recently become available in diagnostics [91]. It is possible to achieve that by training a well-defined deep learning architecture in a Bayesian setting, such as learning a distribution over model parameters. This approach also increases the generalizability of the model that is trained with a relatively small training set [20, 82].

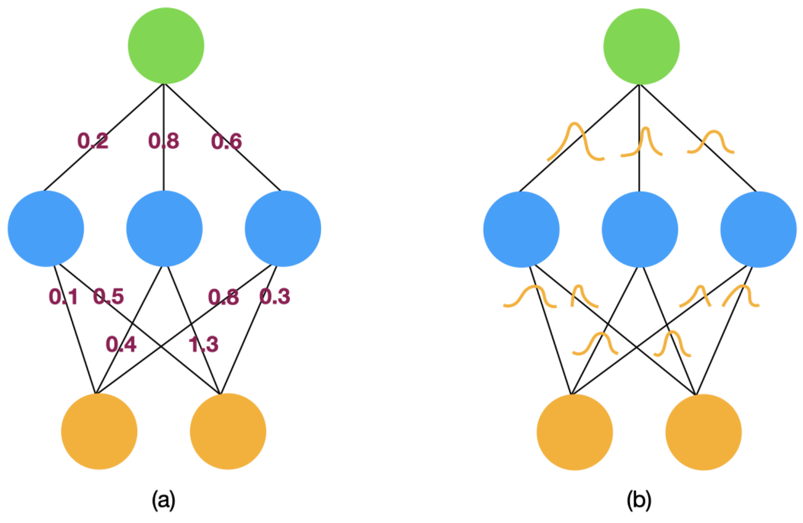


Figure 3.4. Illustration (a) shows a traditional MLP. Illustration (b) shows Bayesian neural network where weights are no longer deterministic values but random variables.

The idea behind BNNs is constructing a MLP with weights and biases that are random variables. Let us first formulate traditional multilayer perceptrons (MLP) (see Section 2.2.1) in a Bayesian setting. Let there be a dataset $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, containing inputs and targets drawn i.i.d. from the true data distribution. The model can be written as a predictive probability distribution over the targets $\mathbf{y}$ given the inputs $\mathbf{x}$ and model parameters $\mathcal{W}$, which are weights and biases of the MLP:

$$p(\mathbf{y} \mid \mathbf{x}, \mathcal{W}). \tag{3.47}$$

In order to learn weights of this model, it is possible to use Bayes theorem:

$$p(\mathcal{W} \mid \mathbf{x}, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x}, \mathcal{W}) p(\mathcal{W})}{p(\mathbf{y} \mid \mathbf{x})}. \tag{3.48}$$

In Equation (3.48), the posterior distribution is not tractable. It is not possible to compute the evidence, $p(\mathbf{y} \mid \mathbf{x}) = \int p(\mathbf{y} \mid \mathbf{x}, \mathcal{W}) p(\mathcal{W}) d\mathcal{W}$, because it is a high dimensional integral due to the large number of parameters $\mathcal{S}$.

Therefore, it is possible to approximate the posterior in Equation (3.48) by using amortized variational inference methods. Similar to the ELBO term that is derived for the VAE (see Equation (3.13)), it is possible to write an ELBO term to be maximized with respect the parameters of the approximate posterior $q_\phi(\mathcal{W})$:

$$\mathcal{L}_\phi(\mathbf{y}) = \mathbb{E}_{q_\phi(\mathcal{W})}\left[p(\mathbf{y} \mid \mathbf{x}, \mathcal{W})\right] - \mathbb{E}_{q_\phi(\mathcal{W})}\left[\log \frac{q_\phi(\mathcal{W})}{p(\mathcal{W})}\right]. \tag{3.49}$$

The prior $p(\mathcal{W})$ is generally selected as a standard Gaussian distribution $\mathcal{N}(0, I)$. The model can be optimized by using minibatch Stochastic Gradient Descent [23]. Therefore, the reparameterization trick (see Equation (3.23)) is used in order to back-propagate through the random variable $\mathcal{W}$. After the posterior is approximated, the model can be used for prediction by marginalizing out the network parameters given the test data $\mathbf{x}^*, \mathbf{y}^*$ [92]:

$$p(\mathbf{y}^* \mid \mathbf{x}^*) = \int p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{W}) q_\phi(\mathcal{W}) d\mathcal{W}. \tag{3.50}$$

It is vital to highlight that MLPs are not stochastic models, which means that they do not directly assign probabilities to their outputs. This is why the predictive distribution in Equation (3.50) must be summarized by mean and variance statistics using Monte-Carlo sampling. Given the MLP, a neural network with parameters $\mathcal{W}$ ($NN_\mathcal{W}$), the average model prediction can be computed by model averaging [92]:

$$\mathcal{W} \sim q_\phi(\mathcal{W}) \tag{3.51}$$

$$\hat{\mathbf{y}} = \mathbb{E}_{q_\phi(\mathcal{W})}\left[NN_\mathcal{W}(\mathbf{x}^*)\right] = \frac{1}{N}\sum_{i=1}^{N} NN_{\mathcal{W}_i}(\mathbf{x}^*). \tag{3.52}$$

In order to quantify the model uncertainty, it is possible to compute the empirical covariance matrix:

$$\begin{aligned}\Sigma_{\mathbf{y}^*|\mathbf{x}^*} &= \mathbb{E}_{q_\phi(\mathcal{W})}\left[(NN_{\mathcal{W}}(\mathbf{x}^*) - \hat{\mathbf{y}})(NN_{\mathcal{W}}(\mathbf{x}^*) - \hat{\mathbf{y}})^\top\right] \\ &= \frac{1}{N-1}\sum_{i=1}^{N}(NN_{\mathcal{W}_i}(x^*) - \hat{y})(NN_{\mathcal{W}_i}(\mathbf{x}^*) - \hat{\mathbf{y}})^\top. \end{aligned} \tag{3.53}$$

It is possible to build bridges between BNNs and ensemble learning. Ensemble learning aims to train different models, to create ensemble models and use them for generating predictions [93]. For instance, it is possible to use the mean of models' predictions for a regression task or to use other fusion strategies such as voting for a classification task. Learning a distribution over weights creates an infinite ensemble of neural networks, where Equations (3.52) and (3.53) are equivalent approximating the output and uncertainty of the ensemble, respectively [20].

3.5. ODE2VAE

Ordinary Differential Equation Variational Auto-Encoder (ODE2VAE) proposes a second order latent ODE model [5]. The model has the objective of learning latent dynamics of high-dimensional sequential data such as videos. It utilizes VAEs [1] and Neural ODEs [17] for encoding the frames and modeling the continuous dynamics in the latent space, respectively. In other words, ODE2VAE models the latent dynamics by using a continuous-time probabilistic ODE. The continuous latent representations are called latent trajectories [5].

Using second order latent ODEs enables the decomposition of latent trajectories in a hierarchical manner. The hierarchical decomposition of the latent space imposes an inductive bias to the model with physical intuitions. The model has three decomposed and continuous latent trajectories: latent position, velocity, and acceleration trajectories.

The latent position trajectory carries the latent representations that are used for decoding the video frames. The latent velocity trajectory drives the latent position trajectory through a first-order latent ODE. The latent acceleration trajectory governs the latent velocity trajectory through another first-order latent ODE. Therefore, the ODE2VAE model infers continuous latent dynamics of a sequential input by using these coupled latent trajectories.

The latent dynamics and underlying generative process for the sequential data $\mathbf{x}_{0:N}$ with $N+1$ frames at time points $0:T$ can be summarized as [5]:

$$\mathbf{s}_0 \sim p(\mathbf{s}_0) \tag{3.54}$$

$$\mathbf{v}_0 \sim p(\mathbf{v}_0) \tag{3.55}$$

$$\mathbf{s}_t = \mathbf{s}_0 + \int_0^t \mathbf{v}_\tau d\tau \tag{3.56}$$

$$\mathbf{v}_t = \mathbf{v}_0 + \int_0^t \mathbf{f}^*(\mathbf{s}_\tau, \mathbf{v}_\tau) d\tau \tag{3.57}$$

$$\mathbf{x}_i \sim p(\mathbf{x}_i \mid \mathbf{s}_i) \tag{3.58}$$

where $p(\mathbf{s}_0)$ and $p(\mathbf{v}_0)$ denote true posterior distributions, $\mathbf{s}_0$ and $\mathbf{v}_0$ correspond to the initial latent states in the latent position and velocity spaces, $\mathbf{s}_t$ denotes the latent position at time t and is driven by the differential field of $\mathbf{v}_t$, $\mathbf{v}_t$ denotes the latent velocity at time t and is driven by the acceleration field $\mathbf{f}^*(\mathbf{s}_\tau, \mathbf{v}_\tau)$, and $p(\mathbf{x}_i \mid \mathbf{s}_i)$ denotes the likelihood which is used to decode the frame $\mathbf{x}_i$.

Due to the intractability issues, the ODE2VAE model aims to learn parameters for modeling the generative process by using amortized variational inference methods [35, 40] (see Section 3.1). It is best to derive the approximate posterior as we unfold the model itself.

The model has two encoders parameterized by convolutional neural networks that project input frames to position and velocity latent spaces:

$$(\boldsymbol{\mu}_{\mathbf{s}}, \log \boldsymbol{\sigma}_{\mathbf{s}}) = \mathcal{E}_{\texttt{pos}}(\mathbf{x}_0) \tag{3.59}$$

$$(\boldsymbol{\mu}_{\mathbf{v}}, \log \boldsymbol{\sigma}_{\mathbf{v}}) = \mathcal{E}_{\texttt{vel}}(\mathbf{x}_{0:m}) \tag{3.60}$$

$$q_{\texttt{pos}}(\mathbf{s}_0 \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{s}_0; \boldsymbol{\mu}_{\mathbf{s}}, \operatorname{diag}(\boldsymbol{\sigma}_{\mathbf{s}})) \tag{3.61}$$

$$q_{\texttt{vel}}(\mathbf{v}_0 \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{v}_0; \boldsymbol{\mu}_{\mathbf{s}}, \operatorname{diag}(\boldsymbol{\sigma}_{\mathbf{v}})) \tag{3.62}$$

where the approximate posterior distribution is denoted by q, the position encoder $\mathcal{E}_{\texttt{pos}}$ has the input of the first frame, the velocity encoder $\mathcal{E}_{\texttt{vel}}$ has first m frames as its input, m is called amortized inference length. Both approximate posteriors follow multivariate Gaussian distributions parameterized by the outputs of the encoders. For a clear notation, it is possible to combine the approximate posteriors into a single multivariate Gaussian distribution with random variable $\mathbf{z}_t$:

$$\mathbf{z}_t = \begin{bmatrix} \mathbf{s}_t \\ \mathbf{v}_t \end{bmatrix}. \tag{3.63}$$

The variational approximation for the posterior distribution becomes:

$$\begin{aligned} q_{\text{enc}}\left(\mathbf{z}_0 \mid \mathbf{x}_{0:N}\right) &= q_{\text{enc}}\left(\begin{pmatrix} \mathbf{s}_0 \\ \mathbf{v}_0 \end{pmatrix} \mid \mathbf{x}_{0:N}\right) \\ &= \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_{\mathbf{s}}(\mathbf{x}_0) \\ \boldsymbol{\mu}_{\mathbf{v}}(\mathbf{x}_{0:m}) \end{pmatrix}, \begin{pmatrix} \operatorname{diag}\left(\boldsymbol{\sigma}_{\mathbf{s}}(\mathbf{x}_0)\right) & \mathbf{0} \\ \mathbf{0} & \operatorname{diag}\left(\boldsymbol{\sigma}_{\mathbf{v}}(\mathbf{x}_{0:m})\right) \end{pmatrix}\right) \end{aligned} \tag{3.64}$$

denotes amortized approximate posterior for the initial latent state, which is a combined state of the initial latent position and velocity states.

After the approximate posterior of the initial latent state is modeled, the next step approximates the posterior for the latent dynamics, which models future latent states by using the acceleration field.

It is possible to write down the second order latent ODE by using two first-order coupled ODEs [5]:

$$\underbrace{\begin{bmatrix} \mathbf{s}_t \\ \mathbf{v}_t \end{bmatrix}}_{\mathbf{z}_t} = \begin{bmatrix} \mathbf{s}_0 \\ \mathbf{v}_0 \end{bmatrix} + \int_0^t \underbrace{\begin{bmatrix} \mathbf{v}_\tau \\ \mathbf{f}_\mathcal{W}\left(\mathbf{s}_\tau, \mathbf{v}_\tau\right) \end{bmatrix}}_{\tilde{\mathbf{f}}_\mathcal{W}(\mathbf{s}_\tau, \mathbf{v}_\tau)} d\tau \tag{3.65}$$

$$\dot{\mathbf{s}}_t = \mathbf{v}_t \qquad \dot{\mathbf{v}}_t = \mathbf{f}_\mathcal{W}\left(\mathbf{s}_t, \mathbf{v}_t\right) \tag{3.66}$$

where $\mathbf{f}_\mathcal{W}(\mathbf{s}_t, \mathbf{v}_t)$ denotes the acceleration field.

The next step is approximating the posterior of the acceleration field, which drives the latent dynamics. The true acceleration field, $f^*(\mathbf{s}_t, \mathbf{v}_t)$, is modeled by using a Bayesian neural network $\mathbf{f}_\mathcal{W}(\mathbf{s}_t, \mathbf{v}_t)$ [5]. The posterior for the BNN is over the model parameters $\mathcal{W}$, which are assumed to be in the form:

$$q(\mathcal{W}) = \mathcal{N}(\mathcal{W}; \boldsymbol{\mu}_\mathbf{f}, \mathrm{diag}(\boldsymbol{\sigma}_\mathbf{f})). \tag{3.67}$$

The last variational approximation is for the latent dynamics of the system, which gives a distribution over the latent states following the initial latent state and the acceleration field. The latent dynamics is modeled by using continuous normalizing flows [17] (see Equations (3.44) and (3.45)). The differential field that governs logarithm of the approximate posterior can be derived by using instantaneous change of variables theorem following Equation (3.65):

$$\begin{aligned} \frac{\partial \log q\left(\mathbf{z}_t \mid \mathcal{W}\right)}{\partial t} &= -\operatorname{Tr}\left(\frac{d\tilde{\mathbf{f}}_\mathcal{W}\left(\mathbf{z}_t\right)}{d\mathbf{z}_t}\right) dt \\ &= -\operatorname{Tr}\begin{pmatrix} \frac{\partial \mathbf{v}_t}{\partial \mathbf{s}_t} & \frac{\partial \mathbf{v}_t}{\partial \mathbf{v}_t} \\ \frac{\partial \mathbf{f}_\mathcal{W}(\mathbf{s}_t, \mathbf{v}_t)}{\partial \mathbf{s}_t} & \frac{\partial \mathbf{f}_\mathcal{W}(\mathbf{s}_t, \mathbf{v}_t)}{\partial \mathbf{v}_t} \end{pmatrix} \\ &= -\operatorname{Tr}\left(\frac{\partial \mathbf{f}_\mathcal{W}\left(\mathbf{s}_t, \mathbf{v}_t\right)}{\partial \mathbf{v}_t}\right). \end{aligned} \tag{3.68}$$

The last step follows the fact $\frac{\partial \mathbf{v}_t}{\partial \mathbf{s}_t} = 0$. Therefore, the dynamics of the variational log densities can be written as [5]:

$$\log q\left(\mathbf{z}_T \mid \mathcal{W}\right) = \log q\left(\mathbf{z}_0 \mid \mathcal{W}\right) - \int_0^T \operatorname{Tr}\left(\frac{\partial \mathbf{f}_{\mathcal{W}}\left(\mathbf{s}_\tau, \mathbf{v}_\tau\right)}{\partial \mathbf{v}_\tau}\right) d\tau \tag{3.69}$$

where $\mathbf{z}_t = [\mathbf{s}_t, \mathbf{v}_t]$.

Now, we are ready to write down the factorized approximate posterior by using the variational approximations derived previously:

$$q\left(\mathcal{W}, \mathbf{z}_{0:N} \mid \mathbf{x}_{0:N}\right) = q(\mathcal{W}) q_{\text{enc}}\left(\mathbf{z}_0 \mid \mathbf{x}_{0:N}\right) q_{\text{ode}}\left(\mathbf{z}_{1:N} \mid \mathbf{z}_0, \mathcal{W}\right) \tag{3.70}$$

where the factorized approximate posterior has three components, which are for the BNN, encoder, and latent dynamics. The ELBO term, which is the lower bound of the marginal log-likelihood, can be written as [5]:

$$\begin{aligned}
\log p(X) \geq\ & \underbrace{\mathcal{L}_{\text{ODE2VAE}}(X)}_{\text{ELBO}} \\
=\ & \underbrace{-\mathcal{D}_{KL}[q(\mathcal{W}, Z \mid X) || p(\mathcal{W}, Z)]}_{\text{Regularization}} + \underbrace{\mathbb{E}_{q(\mathcal{W}, Z \mid X)}[\log p(X \mid \mathcal{W}, Z)]}_{\text{Reconstruction}} \\
=\ & -\mathbb{E}_{q(\mathcal{W}, Z \mid X)}\left[\log \frac{q(\mathcal{W}) q(Z \mid \mathcal{W}, X)}{p(\mathcal{W}) p(Z)}\right] + \mathbb{E}_{q(\mathcal{W}, Z \mid X)}[\log p(X \mid \mathcal{W}, Z)] \\
=\ & -\mathcal{D}_{KL}[q(\mathcal{W}) \| p(\mathcal{W})] \\
& + \mathbb{E}_{q(\mathcal{W}, Z \mid X)}\left[-\log \frac{q(Z \mid \mathcal{W}, X)}{p(Z)} + \log p(X \mid \mathcal{W}, Z)\right] \\
=\ & -\underbrace{\mathcal{D}_{KL}[q(\mathcal{W}) \| p(\mathcal{W})]}_{\text{ODE regularization}} + \underbrace{\mathbb{E}_{q_{\text{enc}}}\left[-\log \frac{q_{\text{enc}}\left(\mathbf{z}_0 \mid X\right)}{p\left(\mathbf{z}_0\right)} + \log p\left(\mathbf{x}_0 \mid \mathbf{z}_0\right)\right]}_{\text{VAE loss}} \\
& + \underbrace{\sum_{i=1}^N \mathbb{E}_{q_{\text{enc}}}\left[\mathbb{E}_{q_{\text{ode}}}\left[-\log \frac{q_{\text{ode}}\left(\mathbf{z}_i \mid \mathbf{z}_0, \mathcal{W}\right)}{p\left(\mathbf{z}_i\right)} + \log p\left(\mathbf{x}_i \mid \mathbf{z}_i\right)\right]\right]}_{\text{Dynamic loss}} .
\end{aligned} \tag{3.71}$$

In Equation (3.71), X and Z denote the sequences $\mathbf{x}_{0:N}$ and $\mathbf{z}_{0:N}$, and the expectations are computed with respect to the corresponding variational posteriors denoted in Equation (3.70). The priors $p(\mathcal{W})$, $p(\mathbf{z}_0)$, and $p(\mathbf{z}_{1:N})$ are standard Gaussians. In Equation (3.71), the first term regularizes the BNN by regularizing the network weights that are used to model the acceleration field. The second term is in the form of a standard ELBO term for a VAE (see Equation (3.13)). The last term corresponds to an ELBO term for the latent sequence, which is driven by the latent ODE (see Equation (3.65)). This is why the expectation is taken over the variational approximation for the latent ODE. With the same training procedure with VAEs, the ELBO term is optimized w.r.t. weights of the BNN, parameters of the encoder and decoder networks by using mini-batches [5].

[5] provides a penalized ELBO formalization that offers a more stable training of the model. The intuition is scaling the components of the ELBO in Equation (3.24) to balance their contributions to objective function during the course of optimization. The penalized version of the ELBO is:

$$\begin{aligned}\mathcal{L}_{\text{ODE2VAE}^*} = & -\beta_{\mathcal{W}}\mathcal{D}_{KL}[q(\mathcal{W})||p(\mathcal{W})] \\ & + \mathbb{E}_{q(\mathcal{W},Z|X)}\left[-\log\frac{q(Z \mid \mathcal{W}, X)}{p(Z)} + \log p(X \mid \mathcal{W}, Z)\right] \\ & - \gamma\mathbb{E}_{q(\mathcal{W})}\left[\mathcal{D}_{KL}\left[q_{\text{ode}}(Z \mid \mathcal{W}, X)\|q_{\text{enc}}(Z \mid X)\right]\right] \end{aligned} \tag{3.72}$$

where $\beta_W = |a|/|\mathcal{W}|$ is chosen as the ratio of the latent space dimension a and the number of weights in BNN. It is noted that the penalized ELBO objective helps to balance the contributions of penalties over $\mathcal{W}$ and $\mathbf{z}_i$ [5]. Additionally, it introduces another regularization term that aims to prevent underfit of the encoder due to the dominance of dynamic loss generated by the long output sequences [5]. It minimizes the KL distance between the density modeled by the latent ODE and density generated by the encoder. Hence, the encoder is also trained with whole sequence. The γ term is chosen by cross-validation [5].

The decoder is parameterized by a transposed convolutional neural network. It takes the latent position states and outputs parameters for a Bernoulli distribution over pixels of the output given that the inputs are a sequence of grayscale images:

$$\mathbf{p} = \mathcal{D}_{\text{pos}}(\mathbf{s_t}) \tag{3.73}$$

$$p(\mathbf{x}_t \mid \mathbf{s}_t) = \mathcal{B}(\mathbf{x}_t; \mathbf{p}) \tag{3.74}$$

where $\mathcal{D}$ is the decoder network and $\mathcal{B}(\mathbf{x}; \mathbf{p})$ is a Bernoulli distribution (see Equation (3.21)). In Figure 3.5, one can find a illustrative scheme of the ODE2VAE model, which shows how the model's components interact with each other.

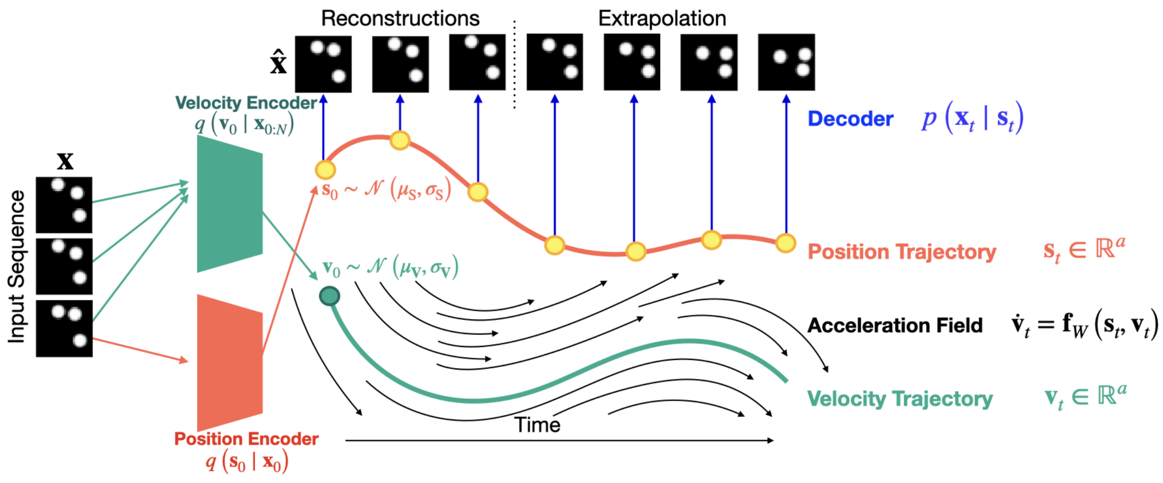

Reconstructions
Extrapolation
$\hat{\mathbf{x}}$
Velocity Encoder
$q(\mathbf{v}_0 \mid \mathbf{x}_{0:N})$
$\mathbf{x}$
Input Sequence
$\mathbf{s}_0 \sim \mathcal{N}(\mu_S, \sigma_S)$
$\mathbf{v}_0 \sim \mathcal{N}(\mu_V, \sigma_V)$
Decoder $p(\mathbf{x}_t \mid \mathbf{s}_t)$
Position Trajectory $\mathbf{s}_t \in \mathbb{R}^a$
Acceleration Field $\dot{\mathbf{v}}_t = \mathbf{f}_W(\mathbf{s}_t, \mathbf{v}_t)$
Velocity Trajectory $\mathbf{v}_t \in \mathbb{R}^a$
Time
Position Encoder
$q(\mathbf{s}_0 \mid \mathbf{x}_0)$

Figure 3.5. Illustration of the ODE2VAE model. The figure is adapted from [5].

4. β-ODE2VAE AND ODE2VAE-c

The β-ODE2VAE model combines ideas of the β-VAEs [18] and ODE2VAE [5]. The motivation of the β-ODE2VAE is modeling the latent dynamics in a regularized manner, which may lead to approximately disentangled and interpretable latent representations.

A video is a sequence of frames that carries spatiotemporal information through the pixels of its frames. There are static and dynamic components of a video. The static components are shared by all the frames in the sequence, whereas the dynamic components may be updated in each frame. For instance, if there are n bouncing balls confined in a 2D frame, the balls' count, shape, radius, mass are the static components, their position, velocity, and acceleration are the dynamic components. Learning the latent dynamics of a video through a generative model deals with both the static and dynamic latent variables. If a generative model learns the static and dynamic latent variables separately, it disentangles the time invariant and time dependent latent features [59].

The ODE2VAE model does not have an explicit architectural choice, regularization, or assumption for disentangling static and dynamic components [5]. However, we show that it is possible to attempt learning disentangled representations of a video by regularizing the ODE2VAE model. We take advantage of the hierarchical latent spaces that latent second order ODE imposes over the ODE2VAE model (see Figure 3.5 for ODE2VAE illustration).

As summarized in Section 3.5, the ODE2VAE model reconstructs and extrapolates future frames by decoding time steps of the position latent trajectory. Let us assume that position and velocity latent spaces are each a dimensional, $\mathbf{s}_t \in \mathbb{R}^a$ and $\mathbf{v}_t \in \mathbb{R}^a$.

Since the static components of a sequence are constant over the sequence, there should exist latent units that capture static components, which are the same for all time steps along the continuous latent trajectory. This property may be fulfilled by putting an additional constraint over the velocity latent variable $\mathbf{v}_t$ at time t. The velocity latent variable denotes a differential field over the corresponding units of the position latent variable. Therefore, if the velocity latent variable has units with values that are close to zero, this means there is a minimal change on the corresponding units of the position latent variable. When this property holds over all time steps, it constrains the position latents to have approximate static components. Let us assume that there are $a' < a$ units of the velocity latent variable that have values close to zero. As a result, there should be $a' < a$ static components that are shared among all time points over the latent position trajectory. For example, if $a = 3$ and $a' = 1$, the position and velocity latent spaces has a dimensions $s_t, v_t \in \mathbb{R}^3$ and there is at least a single static component which is not perturbed as the other position latent variables due to the velocity latent variable with a value close to zero:

$$\underbrace{\begin{bmatrix} {s_t}^1 \\ {s_t}^2 \\ {s_t}^3 \end{bmatrix}}_{\mathbf{s}_t} = \underbrace{\begin{bmatrix} {s_0}^1 \\ {s_0}^2 \\ {s_0}^3 \end{bmatrix}}_{\mathbf{s}_0} + \int_0^t \underbrace{\begin{bmatrix} {v_\tau}^1 \\ {v_\tau}^2 \\ {v_\tau}^3 \approx 0 \end{bmatrix}}_{\dot{\mathbf{s}}_t = \mathbf{v}_t} d\tau \tag{4.1}$$

where ${s_t}^3 \approx {s_0}^3$.

To enforce this property, it may be necessary to regularize the position and velocity latents independently. Therefore, the first step is separating the combined latent states $\mathbf{z}_t$ of ODE2VAE. Then, it is possible to re-write the logarithm of the approximate posterior for position and velocity latents by using the instantaneous change of variables theorem:

$$\frac{\partial \log q\left(\mathbf{s}_t \mid \mathcal{W}\right)}{\partial t} = -\operatorname{Tr}\left(\frac{d\mathbf{v}_t}{d\mathbf{s}_t}\right) dt = -\operatorname{Tr}\left(\frac{\partial \mathbf{v}_t}{\partial \mathbf{s}_t}\right) = 0 \tag{4.2}$$

$$\frac{\partial \log q\left(\mathbf{v}_t \mid \mathcal{W}\right)}{\partial t} = -\operatorname{Tr}\left(\frac{d\mathbf{f}_{\mathcal{W}}\left(\mathbf{s}_t, \mathbf{v}_t\right)}{d\mathbf{s}_t}\right) dt = -\operatorname{Tr}\left(\frac{\partial \mathbf{f}_{\mathcal{W}}\left(\mathbf{s}_t, \mathbf{v}_t\right)}{\partial \mathbf{v}_t}\right). \tag{4.3}$$

Therefore, the dynamics of the variational log densities becomes:

$$\log q\left(\mathbf{s}_T \mid \mathcal{W}\right) = \log q\left(\mathbf{s}_0 \mid \mathcal{W}\right) - \cancelto{0}{\int_0^T 0\, d\tau} \tag{4.4}$$

$$\log q\left(\mathbf{v}_T \mid \mathcal{W}\right) = \log q\left(\mathbf{v}_0 \mid \mathcal{W}\right) - \int_0^T \operatorname{Tr}\left(\frac{\partial \mathbf{f}_{\mathcal{W}}\left(\mathbf{s}_\tau, \mathbf{v}_\tau\right)}{\partial \mathbf{v}_\tau}\right) d\tau \tag{4.5}$$

which creates separate densities for position and velocity trajectories. The ELBO for the ODE2VAE model can be re-written as:

$$\begin{aligned}
\log p(X) \geq &-\mathcal{D}_{KL}[q(\mathcal{W}) \| p(\mathcal{W})] \\
&+ \mathbb{E}_{q(\mathcal{W}, Z \mid X)}\left[-\log \frac{q(Z \mid \mathcal{W}, X)}{p(Z)} + \log p(X \mid \mathcal{W}, Z)\right] \\
= &-\mathcal{D}_{KL}[q(\mathcal{W}) \| p(\mathcal{W})] + \mathbb{E}_{q(\mathcal{W})}\left[\mathbb{E}_{q(Z \mid \mathcal{W}, X)}\left[-\log \frac{q(Z \mid \mathcal{W}, X)}{p(Z)}\right]\right] \\
&+ \mathbb{E}_{q(\mathcal{W}, Z \mid X)}\left[\log p(X \mid \mathcal{W}, Z)\right] \\
= &-\mathcal{D}_{KL}[q(\mathcal{W}) \| p(\mathcal{W})] - \mathbb{E}_{q(\mathcal{W})}\left[\mathcal{D}_{KL}[q(Z \mid \mathcal{W}, X) \| p(Z)]\right] \\
&+ \mathbb{E}_{q(\mathcal{W}, Z \mid X)}\left[\log p(X \mid \mathcal{W}, Z)\right]
\end{aligned} \tag{4.6}$$

where the first term is regularizing weights of BNN, the second term regularizes latent representations including the initial and following latent states, and the last term is log-likelihood term for reconstructions. We will focus on the second term ($\mathcal{L}_{KL}$) and decompose it (we omit the expectation over BNN weights for readability):

$$\begin{aligned}
\mathcal{L}_{KL} &= -\mathcal{D}_{KL}[q(Z \mid \mathcal{W}, X) \| p(Z)] \\
&= -\mathcal{D}_{KL}[q_{\text{enc}}(z_0 \mid X) q_{\text{ode}}(z_{1:N} \mid z_0, \mathcal{W}) \| p(z_{0:N})]
\end{aligned} \tag{4.7}$$

with the decomposed variational posteriors:

$$q_{\text{enc}}(z_0 \mid X) = q_{\text{enc}}(s_0 \mid X) q_{\text{enc}}(v_0 \mid X) \tag{4.8}$$

$$q_{\text{ode}}(z_{1:N} \mid z_0, \mathcal{W}) = q_{\text{ode}}(s_{1:N} \mid s_0, v_0, \mathcal{W}) q_{\text{ode}}(v_{1:N} \mid s_0, v_0, \mathcal{W}). \tag{4.9}$$

In Equation (4.9), the encoders for initial states refer to position and velocity encoders, and the latent ODEs correspond to the decomposed version of the latent ODE for dynamic position and velocity latents. By using the definition of the KL divergence, $\mathcal{L}_{KL}$ becomes:

$$\mathcal{L}_{KL} = \mathbb{E}_{q_{\text{enc}}(z_0|X)}\mathbb{E}_{q_{\text{ode}}(z_{1:N}|z_0,\mathcal{W})}[-\log \frac{q_{\text{enc}}(s_0, v_0 \mid X) q_{\text{ode}}(s_{1:N}, v_{1:N} \mid s_0, v_0, \mathcal{W})}{p(s_0)p(v_0)p(s_{1:N})p(v_{1:N})}] \quad (4.10)$$

where $\mathbf{z}_t = [\mathbf{s}_t, \mathbf{v}_t]$ and the prior distributions are standard multivariate Gaussians. Since the initial densities for the latent position and velocity do not depend on the next latents, the expectation over latent ODE densities can be marginalized out:

$$\begin{aligned}\mathcal{L}_{KL} &= \mathbb{E}_{q_{\text{enc}}(s_0,v_0|X)}[-\log \frac{q_{\text{enc}}(s_0, v_0 \mid X)}{p(s_0)p(v_0)}] \\ &\quad + \mathbb{E}_{q_{\text{enc}}(s_0,v_0|X)}\mathbb{E}_{q_{\text{ode}}(s_{1:N},v_{1:N}|s_0,v_0,\mathcal{W})}[-\log \frac{q_{\text{ode}}(s_{1:N}, v_{1:N} \mid s_0, v_0, \mathcal{W})}{p(s_{1:N})p(v_{1:N})}]. \quad (4.11)\end{aligned}$$

Plugging the Equation (4.8) and 4.9 in it creates the form:

$$\begin{aligned}\mathcal{L}_{KL} &= \mathbb{E}_{q_{\text{enc}}(s_0|X)}[-\log \frac{q_{\text{enc}}(s_0 \mid X)}{p(s_0)}] + \mathbb{E}_{q_{\text{enc}}(v_0|X)}[-\log \frac{q_{\text{enc}}(v_0 \mid X)}{p(v_0)}] \\ &\quad + \mathbb{E}_{q_{\text{enc}}(s_0,v_0|X)}\mathbb{E}_{q_{\text{ode}}(s_{1:N}|s_0,v_0,\mathcal{W})}[-\log \frac{q_{\text{ode}}(s_{1:N} \mid s_0, v_0, \mathcal{W})}{p(s_{1:N})} \\ &\quad + \mathbb{E}_{q_{\text{enc}}(s_0,v_0|X)}\mathbb{E}_{q_{\text{ode}}(v_{1:N}|s_0,v_0,\mathcal{W})}[-\log \frac{q_{\text{ode}}(v_{1:N} \mid s_0, v_0, \mathcal{W})}{p(v_{1:N})}] \quad (4.12)\end{aligned}$$

which can be written in terms of a summation of decomposed KL divergences:

$$\begin{aligned}\mathcal{L}_{KL} &= -\mathcal{D}_{KL}[q_{\text{enc}}(s_0 \mid X) \| p(s_0)] - \mathcal{D}_{KL}[q_{\text{enc}}(v_0 \mid X) \| p(v_0)] \\ &\quad - \mathbb{E}_{q_{\text{enc}}(s_0,v_0|X)}[\mathcal{D}_{KL}[q_{\text{ode}}(s_{1:N} \mid s_0, v_0, \mathcal{W}) \| p(s_{1:N})]] \\ &\quad - \mathbb{E}_{q_{\text{enc}}(s_0,v_0|X)}[\mathcal{D}_{KL}[q_{\text{ode}}(v_{1:N} \mid s_0, v_0, \mathcal{W}) \| p(v_{1:N})]]. \quad (4.13)\end{aligned}$$

When the components for position and velocity latents are grouped, it becomes:

$$\begin{aligned}\mathcal{L}_{KL} = &\underbrace{-\mathcal{D}_{KL}[q_{\text{enc}}(s_0 \mid X)\|p(s_0)] - \mathbb{E}_{q_{\text{enc}}}[\mathcal{D}_{KL}[q_{\text{ode}}(s_{1:N} \mid s_0, v_0, \mathcal{W})\|p(s_{1:N})]]}_{\text{Latent position regularization}} \\ &\underbrace{-\mathcal{D}_{KL}[q_{\text{enc}}(v_0 \mid X)\|p(v_0)] - \mathbb{E}_{q_{\text{enc}}}[\mathcal{D}_{KL}[q_{\text{ode}}(v_{1:N} \mid s_0, v_0, \mathcal{W})\|p(v_{1:N})]]}_{\text{Latent velocity regularization}}. \quad (4.14)\end{aligned}$$

This shows that the densities of initial and dynamic latent states may be regularized separately for position and velocity trajectories. The KL term in Equation (4.14) can be computed by utilizing Equation (4.4) and 4.5. It is possible to regularize the velocity latents by putting a hyperparameters β that adjust the latent capacity of the position and velocity trajectories:

$$\begin{aligned}\mathcal{L}_{KL-\beta} = &-\beta_s \mathcal{D}_{KL}[q_{\text{enc}}(s_0 \mid X)\|p(s_0)] - \beta_v\, \mathcal{D}_{KL}[q_{\text{enc}}(v_0 \mid X)\|p(v_0)] \\ &- \mathbb{E}_{q_{\text{enc}}(s_0,v_0|X)}[\mathcal{D}_{KL}[q_{\text{ode}}(s_{1:N} \mid s_0, v_0, \mathcal{W})\|p(s_{1:N})]] \\ &- \mathbb{E}_{q_{\text{enc}}(s_0,v_0|X)}[\mathcal{D}_{KL}[q_{\text{ode}}(v_{1:N} \mid s_0, v_0, \mathcal{W})\|p(v_{1:N})]] \quad (4.15)\end{aligned}$$

where all the prior densities are selected as the standard multivariate Gaussian distribution. In Equation (4.15), while $\beta = 1$ corresponds to original ODE2VAE formulation, $\beta > 1$ regularizes the model by forcing it to learn more efficient latent position and velocity representations [18]. The regularization of the position latents through β_s disentangles the position units, which may be helpful for disentangling static and dynamic features of the sequence. Moreover, if it achieves this disentanglement the expected static units mean that corresponding units in the velocity latents should be close to zero (see Equation (4.1)). Additionally, the regularization of the velocity units through β_v term disentangles the velocity units for each time step, but most importantly, the units of the latent velocity have a stronger bottleneck, which forces uninformative units to have values close to the prior. Given the fact that the position trajectory is driven by the velocity trajectory, this regularization can be helpful for leaving the corresponding units of the position not updated as described in Equation (4.1).

The final form of the ELBO term for the $\beta-$ODE2VAE is:

$$\begin{aligned}\log p(X) \geq -\mathcal{D}_{KL}[q(\mathcal{W})\|p(\mathcal{W})] + \mathbb{E}_{q(\mathcal{W})}\left[\mathcal{L}_{KL-\beta}\right] \\ + \mathbb{E}_{q(\mathcal{W},Z|X)}\left[\log p(X \mid \mathcal{W}, Z)\right].\end{aligned} \tag{4.16}$$

Our $\beta-$ODE2VAE model proposes disentangling static and dynamic latent features through regularization. However, it is also possible to learn disentangled content and dynamic latents by designing a proper model [59]. The ODE2VAE-c model has an extra encoder on top of the ODE2VAE architecture. The extra encoder is called static encoder that has the same input as the position encoder. The encoder is a convolutional neural network and it outputs parameters for the approximate posterior for the static latents in a latent space with a^* dimensions:

$$(\boldsymbol{\mu}_{\mathbf{c}}, \log \boldsymbol{\sigma}_{\mathbf{c}}) = \mathcal{E}_{\mathtt{static}}(\mathbf{x}_0) \tag{4.17}$$

$$q_{\mathtt{static}}(\mathbf{C} \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{C}; \boldsymbol{\mu}_{\mathbf{c}}, \mathrm{diag}(\boldsymbol{\sigma}_{\mathbf{c}})) \tag{4.18}$$

where $\mathcal{E}_{\mathtt{static}}$ denotes static encoder and the static latent component is a global variable, which is same at every time step. These latent features are concatenated with position latents after the position trajectory is sampled:

$$\mathbf{s}^*{}_t = \begin{bmatrix} \mathbf{s}_t \\ \mathbf{C} \end{bmatrix} \tag{4.19}$$

where $\mathbf{s}^*{}_t$ denotes updated position state after the concatenation. Then, the concatenated latent position states are used for reconstruction and extrapolation as follows:

$$\mathbf{p} = \mathcal{D}_{\mathrm{pos}^*}(\mathbf{s}^*) \tag{4.20}$$

$$p(\mathbf{x}_t \mid \mathbf{s}^*{}_t) = f(\mathbf{x}_t; \mathbf{p}) \tag{4.21}$$

where $\mathcal{D}_{\mathrm{pos}^*}$ denotes the decoder and $f(\mathbf{x}; \mathbf{p})$ is a Bernoulli distribution (see Equation (3.21)).

The decoder has the same architecture with the decoder in the original ODE2VAE formulation expect its input's dimension (see Equation (3.74)). The ODE2VAE-c has the factorized approximate posterior distribution:

$$q\left(\mathcal{W}, \mathbf{z}_{0:N}, \mathbf{C} \mid \mathbf{x}_{0:N}\right) = q(\mathcal{W}) q_{\text{static}}\left(\mathbf{C} \mid \mathbf{x}_0\right) q_{\text{enc}}\left(\mathbf{z}_0 \mid \mathbf{x}_{0:N}\right) q_{\text{ode}}\left(\mathbf{z}_{1:N} \mid \mathbf{z}_0, \mathcal{W}\right) \quad (4.22)$$

where $\mathbf{z}_t = [\mathbf{s}_t, \mathbf{v}_t]$. The ODE2VAE-c has the ELBO formulation:

$$\begin{aligned}
\log p(X) &\geq \underbrace{\mathcal{L}_{\text{ODE2VAE}-\text{c}}(X)}_{\text{ELBO}} \\
&= -\mathcal{D}_{KL}[q(\mathcal{W}) \| p(\mathcal{W})] + \mathbb{E}_{q_{\text{static}}}\left[-\log \frac{q_{\text{static}}\left(\mathbf{C} \mid X\right)}{p\left(\mathbf{C}\right)}\right] \\
&\quad + \mathbb{E}_{q(\mathbf{z}_0, \mathbf{C} \mid X)}\left[-\log \frac{q_{\text{enc}}\left(\mathbf{z}_0 \mid X\right)}{p\left(\mathbf{z}_0\right)} + \log p\left(\mathbf{x}_0 \mid \mathbf{z}_0, \mathbf{C}\right)\right] \\
&\quad + \sum_{i=1}^{N} \mathbb{E}_{q(\mathbf{z}_0, \mathbf{C} \mid X)}\left[\mathbb{E}_{q_{\text{ode}}}\left[-\log \frac{q_{\text{ode}}\left(\mathbf{z}_i \mid \mathbf{z}_0, \mathcal{W}\right)}{p\left(\mathbf{z}_i\right)} + \log p\left(\mathbf{x}_i \mid \mathbf{z}_i, \mathbf{C}\right)\right]\right] \quad (4.23)
\end{aligned}$$

where $q\left(\mathbf{z}_0, \mathbf{C} \mid X\right)$ denotes $q_{\text{static}}\left(\mathbf{C} \mid \mathbf{x}_0\right) q_{\text{enc}}\left(\mathbf{z}_0 \mid \mathbf{x}_{0:N}\right)$ and the expectations are computed with respect to the corresponding approximate posteriors in Equation (4.23). The reconstructions are computed with updated position latents (see Equation (4.19)). This formulation does not allow using the static features in the latent ODE computation. It can be achieved by concatenating the static features with $\mathbf{z}_t$ to compute the acceleration latents. In that case, the variational posterior over the latent ODE becomes $q_{\text{ode}}\left(\mathbf{z}_{1:N} \mid \mathbf{z}_0, \mathbf{C}, \mathcal{W}\right)$. It is also possible to add a penalization term for learning the static latents similar to the ODE2VAE model. In that case the ELBO in Equation (4.23) has an additional regularization term over the static components:

$$-\gamma \sum_{i=1}^{N} \mathcal{D}_{KL}\left[q_{\text{static}}(\mathbf{C} \mid X) \| q_{\text{static}}(\mathbf{C'}_i \mid X)\right] \quad (4.24)$$

where $q_{\text{static}}(\mathbf{C'}_i \mid X)$ term denotes the outputs of the static encoder at each time point i given the ground truth frames.

5. RESULTS

5.1. Datasets

In this thesis, three synthetic physical motion datasets are used in the experiments. These are bouncing balls, simple pendulum, and projectile motion datasets. For each dynamical system, there are sequences of $32 \times 32 \times 1$ images with pixel values between 0 and 1. These are challenging datasets for the ODE2VAE model, since they capture dynamics of different physical phenomena. Moreover, we create two variants of the bouncing ball motion of the single ball. The first one has bouncing ball motion with different ball radii. The second one has two different objects [94], a square and a heart, that follow the dynamics of a single bouncing ball. If not otherwise indicated, for each dataset, the number of cases are 10000, 500, 500 for training, validation, and test sets. All parameters used in data generation are in International System of Units (SI). We also store the moments of collision or change of direction in order to use them in the analysis.

5.1.1. Bouncing Balls

Bouncing balls dataset [95] is a standard benchmark task for models that aim to learn generative temporal modeling [5, 57]. By using the provided implementation [95], we re-implemented the bouncing balls dataset with multiple variants. The datasets capture dynamics of n balls in a 2D box. There is no friction in the motion of the balls, and all collisions are elastic. The 2D box has a side length of 10.0 m. For the dataset with constant ball radius, the radius and mass of the balls are fixed, at 1.2 m and 1.0 kg, respectively. The dataset with multiple radii has the radii of 0.9 m and 1.5 m. Each ball has a velocity that is randomly sampled from a standard normal distribution. The velocities of the balls in the same sequence are normalized, so that the total kinetic energy is constant over each sequence. The frames in the sequences are separated by one second, and the balls' motion is simulated with 0.5 second resolution.

In Figures 5.1, 5.2, 5.3, 5.4, and 5.5 we present different variants of the bouncing balls datasets. We specify the sequence length as T, the number of balls as n, for the training, validation, and test sets. The dataset with $n = 1, 2, 3$ has 9900, 495, and 495 cases for training, validation and test sets.

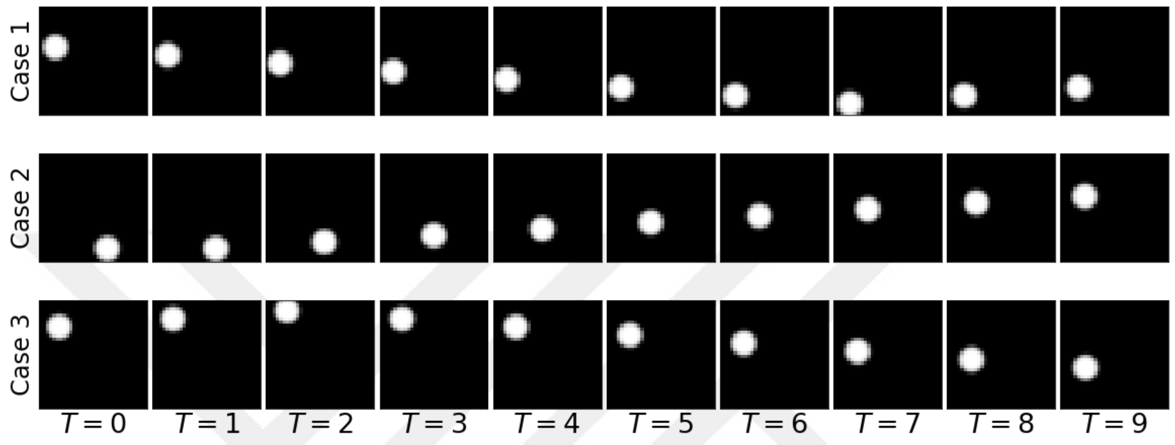


Figure 5.1. Bouncing ball dataset with number of balls $n = 1$ and sequence length $T = 10$.

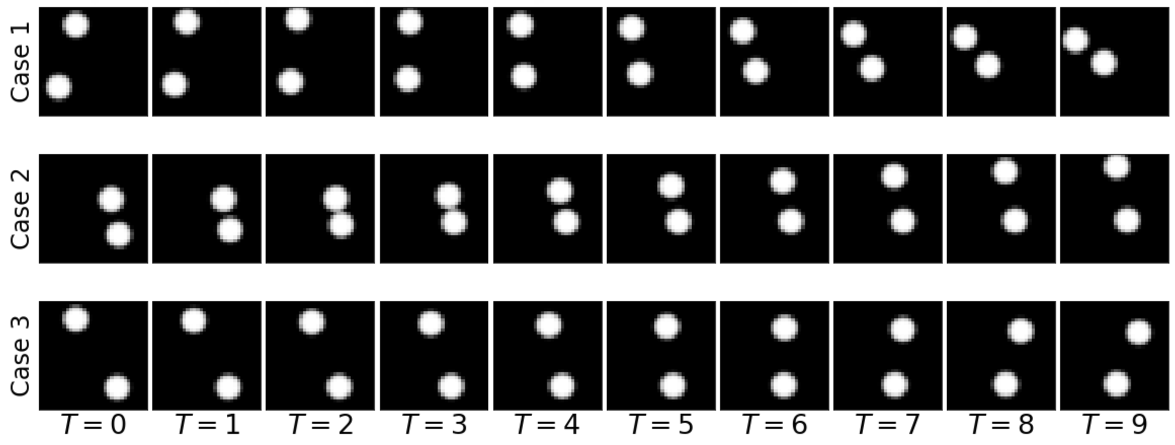


Figure 5.2. Bouncing ball dataset with number of balls $n = 2$ and sequence length $T = 10$.

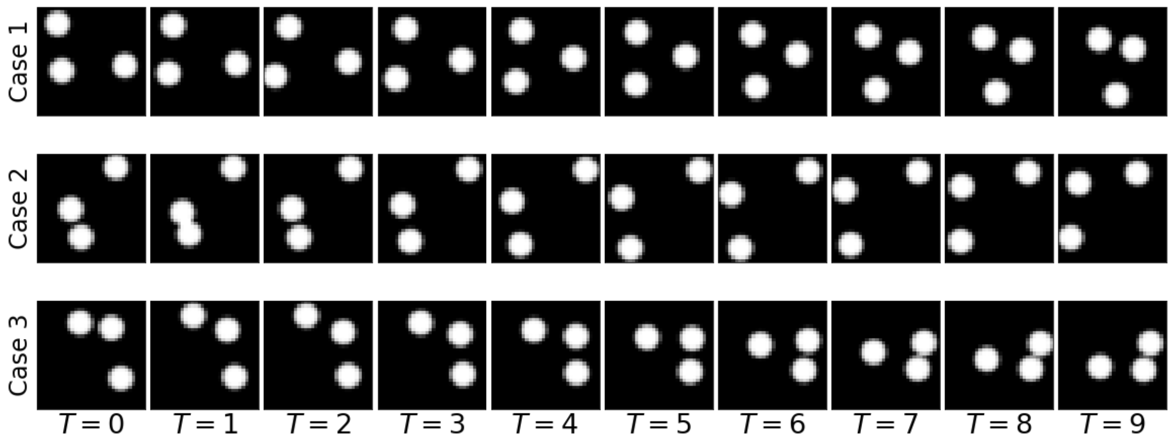


Figure 5.3. Bouncing ball dataset with number of balls $n = 3$ and sequence length $T = 10$.

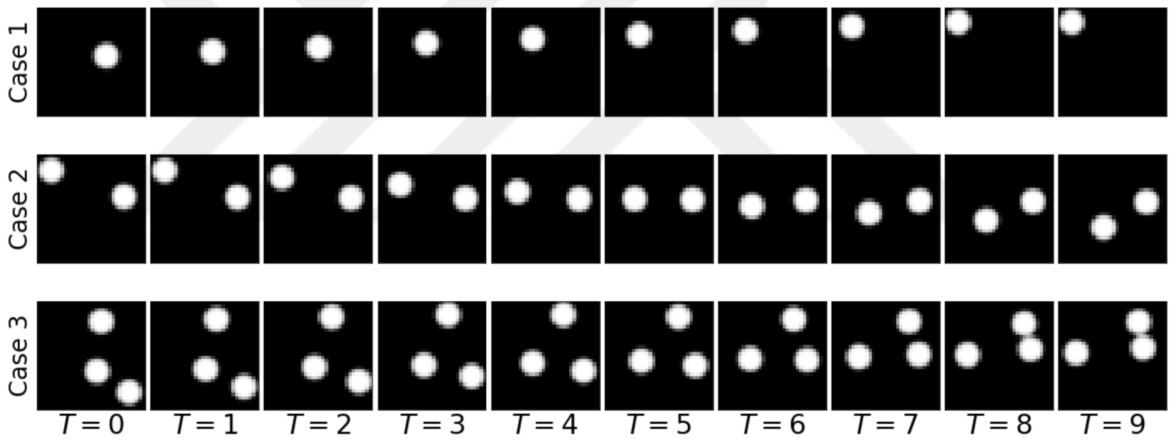


Figure 5.4. Bouncing ball dataset with number of balls $n = 1, 2, 3$, and sequence length $T = 10$. Each value of n has equal number of cases in the dataset.

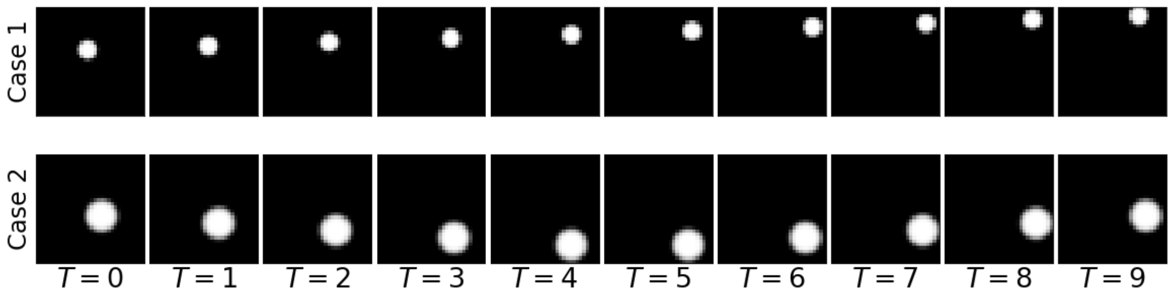


Figure 5.5. Bouncing ball dataset with number of balls $n = 1$, radii of 0.9 and 1.5, and sequence length $T = 10$. For each value of radius there are equal number of cases in the dataset.

5.1.2. Simple Pendulum

A simple pendulum system consists of a point mass, and a pivoted rod with length l, where the mass is suspended from the rod. The mass of the rod is negligible, and there is no air friction in our case. In this thesis, we only consider pendulum motions with small initial angles. Therefore, it is sufficient to model simple pendulum motion. The dynamics of the simple pendulum is approximated by using small angle approximation, $\sin\alpha \approx \alpha$. The dataset captures the periodic motion of the point mass around its equilibrium position. During the motion, the gravitational field is uniform. The object reaches its maximum kinetic energy at its equilibrium position. The magnitude of the restoring force over the object increases as it approaches its highest point of swing. The side length of the 2D square box is 10.0 m. The radius of point mass is 1.0 m. The length of the rod l has a uniform distribution in the range $[3, 6]$. The initial angle for freeing the point mass follows a uniform distribution in the range $[\pi/36, \pi/9]$. The gravitational field has a magnitude of 9.91 $\mathrm{m/s^2}$. The frames are separated by 0.4 seconds, and the motion is simulated by the analytical solution for the simple pendulum motion. Due to the different initial angles, the maximum total kinetic energy for each case is different. Figure 5.6 shows example sequences from the pendulum dataset. Since the aim is to learn the dynamics of the ball, the rod is not visualized in the frames.

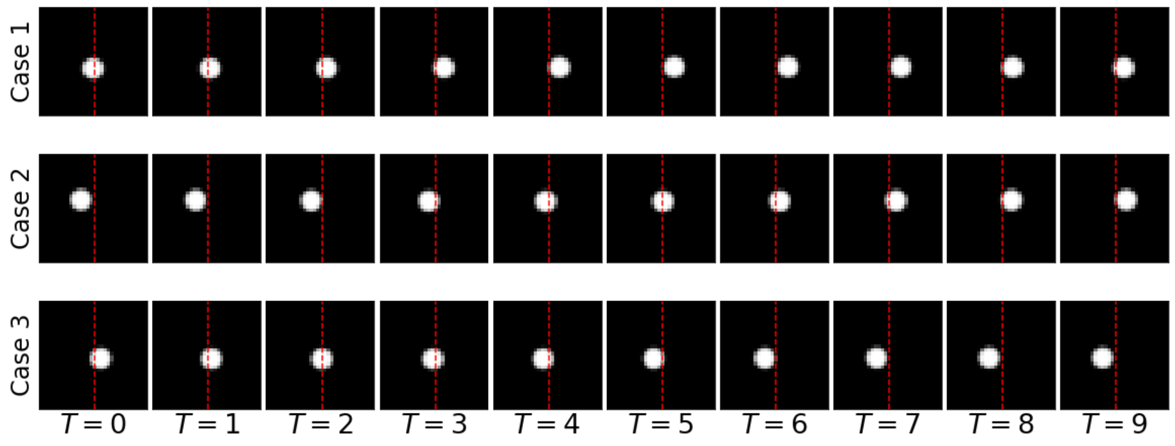


Figure 5.6. Simple pendulum dataset with single point mass hanged from the pivoted rod. The vertical lines denote the equilibrium axis. The sequence length T is 10.

5.1.3. Projectile Motion

The projectile motion dataset consists of a ball, which is projected up in a square frame with the side length of 10.0 m. The ball is affected by gravity and collisions during its motion. The ball reaches its maximum kinetic energy before its first collision with the ground. During the collision it loses some of its kinetic energy. It has a radius of 1.0 m. The initial velocity (m/s) is denoted as $v = [v_x, v_y]$, where v_x and v_y are sampled from a uniform distribution with the ranges $[1, 4]$ and $[0, 1]$. The initial position (m) is denoted as $h = [h_x, h_y]$, where h_x is fixed as zero and h_y has a uniform distribution in the range $[1, 3]$. The coefficient of restitution, which determines the magnitude of the v_y after the ball hits and bounces from the floor, is 0.80. The collision between the ball and floor is assumed to take 0.1 second. The frames are separated by 0.1 second. Figure 5.7 shows example sequences from the projectile motion dataset.

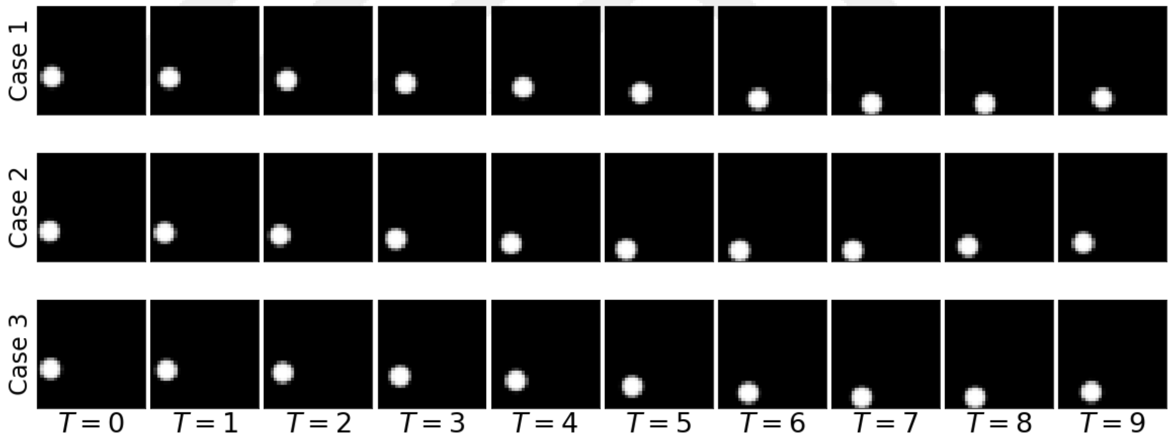


Figure 5.7. Projectile motion dataset with a ball launched from a initial height. The sequence length T is 10.

5.1.4. Bouncing dSprites

The bouncing dSprites dataset captures motion of two type of objects, a square and a heart. Each case has one object which follows the dynamics of a single bouncing ball with the same details described in Section 5.1.1. This dataset is created by merging the dynamics of a bouncing ball with two 2D objects from the dSprites dataset [94]. The frames are separated by a second.

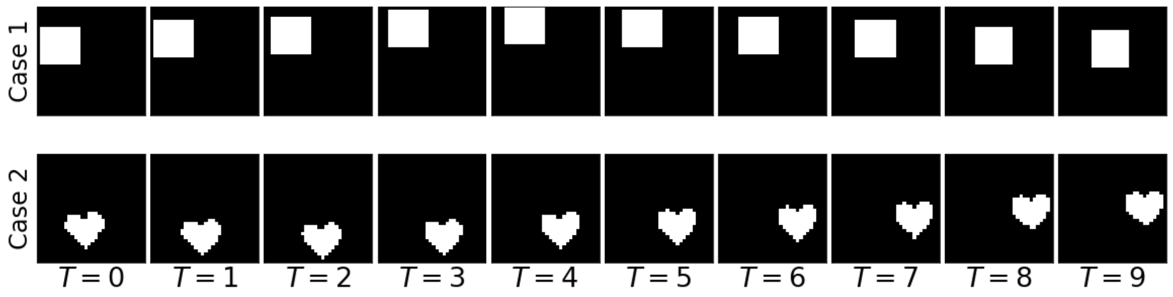


Figure 5.8. Bouncing dSprites dataset with two shapes. For each shape there are equal number of cases in the dataset. The sequence length T is 10.

5.2. Evaluation Metrics

The experiment results are evaluated by using the following quantitative evaluation metrics. The quantitative evaluations are conducted by using the model's latent representations and its reconstructions and extrapolations. As the model first outputs latent representations, its best to mention quantitative metrics that are computed in the latent space.

5.2.1. L2 Norms of the Latent States

Deep latent variable models deal with high dimensional latent states. If the model operates in the time domain, it generates high dimensional latent representations for each time step. L2 (Euclidean) norms of the latent states are used as explanatory and interpretable metrics used in the evaluation of the unsupervised deep generative models that operate with continuous-time data [6, 78].

Since the baseline and proposed models operate with latent second order ODEs, they have the inductive bias of learning the latent dynamics similar to real motion dynamics. The L2 norm of the acceleration field is similar to the magnitude of the force effecting the dynamics in the latent space. Similarly, the L2 norm of the velocity latent variable resembles square root of the total kinetic energy of the system.

5.2.2. Mean Squared Error

Mean squared error (MSE) computes mean of squared errors between true data points and predictions:

$$MSE(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{5.1}$$

where x_i denotes ground truth values, $\hat{x}_i$ denotes predicted values, and n denotes number of data points. The lower MSE represents higher similarity between the ground truth and prediction. When MSE is used with image data, the result may be normalized with respect to the number of pixels of an image, which is called pixel MSE.

5.2.3. Peak Signal-to-Noise Ratio

Peak signal-to-noise ratio (PSNR) measures the quality of the prediction by computing the logarithm of the ratio between the square of the maximum pixel fluctuation among the images and the pixel MSE between the ground truth and predicted image:

$$PSNR(x, \hat{x}) = 10 \log_{10} \left(\frac{{MAX_I}^2}{MSE(x, \hat{x})} \right) \tag{5.2}$$

where ${MAX_I}^2$ is 1 for the grayscale images. As the PSNR score increases, the prediction quality also increases.

5.2.4. Marginal Log-likelihood

Marginal log-likelihood of a data point under the variational model can be computed by using the importance sampling technique [41, 42]:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log \mathbb{E}_{q_\phi} \left[p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) / q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \right]. \tag{5.3}$$

It can be approximated by using a Monte Carlo estimator:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \approx \log \frac{1}{L} \sum_{l=1}^{L} p_{\boldsymbol{\theta}}\left(\mathbf{x}, \mathbf{z}^{(l)}\right) / q_{\boldsymbol{\phi}}\left(\mathbf{z}^{(l)} \mid \mathbf{x}\right) \tag{5.4}$$

$$\mathbf{z}^{(l)} \sim q_{\phi}(\mathbf{z} \mid \mathbf{x}) \tag{5.5}$$

where L denotes the number of samples, $q_{\phi}(\mathbf{z} \mid \mathbf{x})$ denotes the encoder model. As the sample size L gets larger, the approximation becomes a better estimate of the true marginal likelihood [41]. Given the dataset $\mathcal{D}$ with $N_{\mathcal{D}}$ data points, the marginal negative log-likelihood (NLL) can be computed as [41]:

$$-\log p_{\theta}(\mathcal{D}) = -\frac{1}{N_{\mathcal{D}}} \sum_{\mathbf{x} \in \mathcal{D}} \log p_{\theta}(\mathbf{x}). \tag{5.6}$$

5.3. Experiments

We evaluate the performance of the baseline ODE2VAE model [5] on three different physical motion datasets described in Section 5.1. Compared to the baseline model [5], the datasets include variants of the bouncing balls dataset, and two new dynamical settings: simple pendulum and projectile motion. Then, we evaluate the performance of the baseline model on the bouncing balls with $n = 1, 2, 3$, bouncing balls with different radii, and bouncing dSprites datasets. We cannot evaluate the performances of the proposed model variants since their increased complexity alleviates the convergence issues.

5.3.1. Implementation Details

All of the model variants are implemented using the official ODE2VAE model implementation [5] and Tensorflow framework [96]. All of the model variants are trained with Adam Optimizer [25] with the learning rate of $\eta = 0.001$. The models have the learning objective of minimizing the corresponding negative ELBO term.

The amortized inference length is selected as $m = 3$. The learning rate is tuned during the optimization using the exponential learning rate decay of 0.995. During the experiments, the models are trained with the batch size of 32. The number of epochs are specified for each experiment in the corresponding paragraphs. The coefficient γ, for the penalized variational loss function, is chosen as 0.001 as it is suggested in the original work [5]. The network architectures are chosen to be the same as the original ODE2VAE model [5]. The architecture for the ODE2VAE and $\beta-$ODE2VAE is described in Figure A.1 and the ODE2VAE-c's architecture is presented in Figure A.2. All experiments are executed on a single Tesla V100 GPU where each experiment takes approximately two to four days.

5.3.2. Analysis of the Latent States

In this section, we conduct experiments for checking if the baseline ODE2VAE can learn the latent dynamics by preserving physics-motivated quantities and still reconstruct and extrapolate the sequences. We started the experiments with minimum possible number of latent units that correspond to dynamical generating factors ν. This approach has an implicit effect of guiding the model to learn meaningful latent representations since the only way to maximize the ELBO is allocating the latent units by representing the ground truth generating factors. During the analysis, we use the sample size L=10, which is the number of latent states sampled at each time step. Also, we highlight the indices with collisions or direction changes in the related reconstruction figures.

Table 5.1. Performance metrics of the selected model on the bouncing balls dataset with $n = 1$. For the MSE and NLL values, the lower is better. For the PSNR scores, the higher is better. Each metric is computed by using 10 samples per test case.

Model \ Metric	MSE	PSNR	NLL
ODE2VAE, $a = 3$	0.0027 ± 0.0032	28.2601 ± 4.6264	42.5142

The baseline model with the latent dimensionality $a = 2$ cannot capture the dynamics of the motion of the single bouncing ball after it is trained for 250 epochs. We increase the latent dimensionality to $a = 3$ and $a = 12$ and train the models for 250 epochs. Both of the models have captured the dynamics of the single bouncing ball. In Table 5.1, we only report the metrics for model with $a = 3$. In Figures 5.9 and 5.10, we plot MSE and PSNR values for the test cases over the time steps. The model is able to capture physically meaningful latent representations. We present an example case in Figure 5.11. When the ball hits the wall, there is a spike in the norm of the latent acceleration. It can be seen that the standard deviation of the norm of the acceleration field also increases during the collision. This indicates the fact that the output of the BNN has a greater uncertainty during the collision. Some possible reasons for the high uncertainty over the norm of the acceleration latents may be the non-linear motion during the collision and the scarcity of the time steps with collision. The norm of the latent velocity is not changed during the motion in the example case, which suggests the fact that the latent velocity preserves its norm, but changes the direction during the collisions (see Figure B.1 for a detailed example). Figures 5.12a and 5.12b summarize the statistics for the norm of the acceleration field and latent velocity over all test cases with a breakdown for the time steps with and without collision. Figure 5.12a shows that the model generates an acceleration field with a greater magnitude during the collisions. At the collision moments, the high standard deviation over the norm of the acceleration field is meaningful since it depends on the amount of change in the momentum, which varies in the test cases. When there no collision, the model generates an acceleration field with a smaller norm compared to the mean magnitude at the collision time points, which is physically plausible. We note that the real dynamics require the model to generate zero acceleration field when there is no collision. Figure 5.12b shows that the model has increased the magnitude of the velocity latent without collision. Its magnitude decreases during the collision moments. This is not an expected behavior since the ball's stationary moments during the collisions are ignored in the dataset. Therefore, it can be said the model cannot preserve a constant norm of the latent velocity.

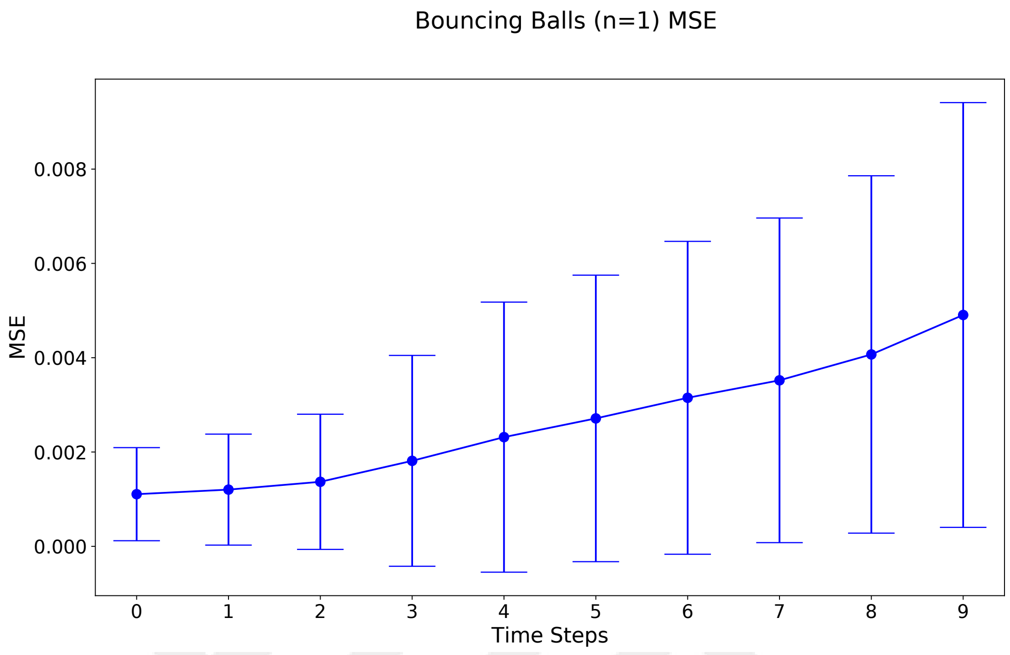


Figure 5.9. MSE values for the bouncing balls dataset with $n = 1$.

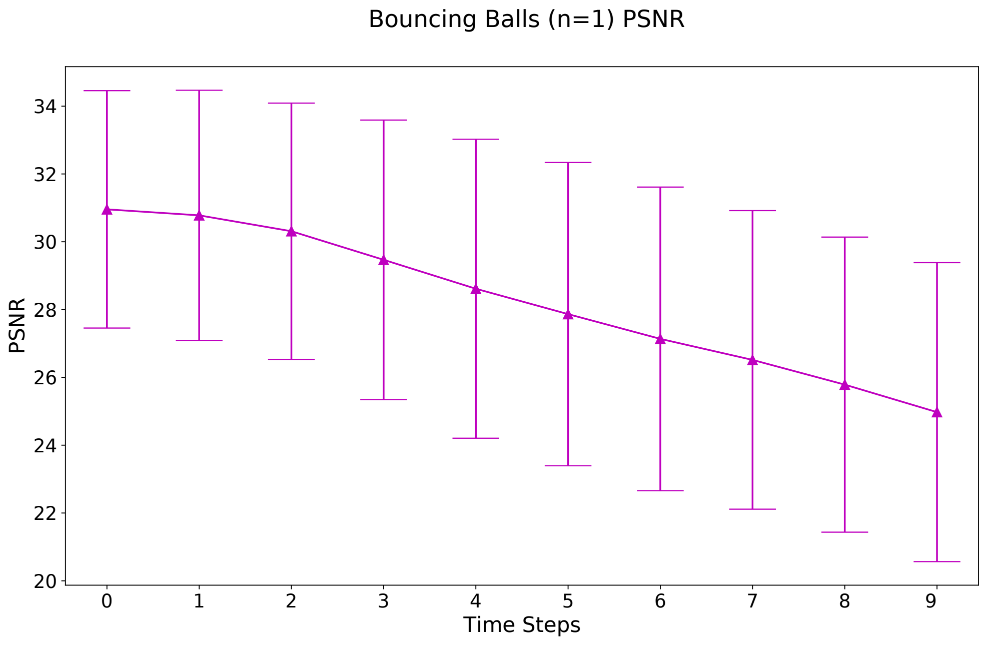


Figure 5.10. PSNR values for the bouncing balls dataset with $n = 1$.

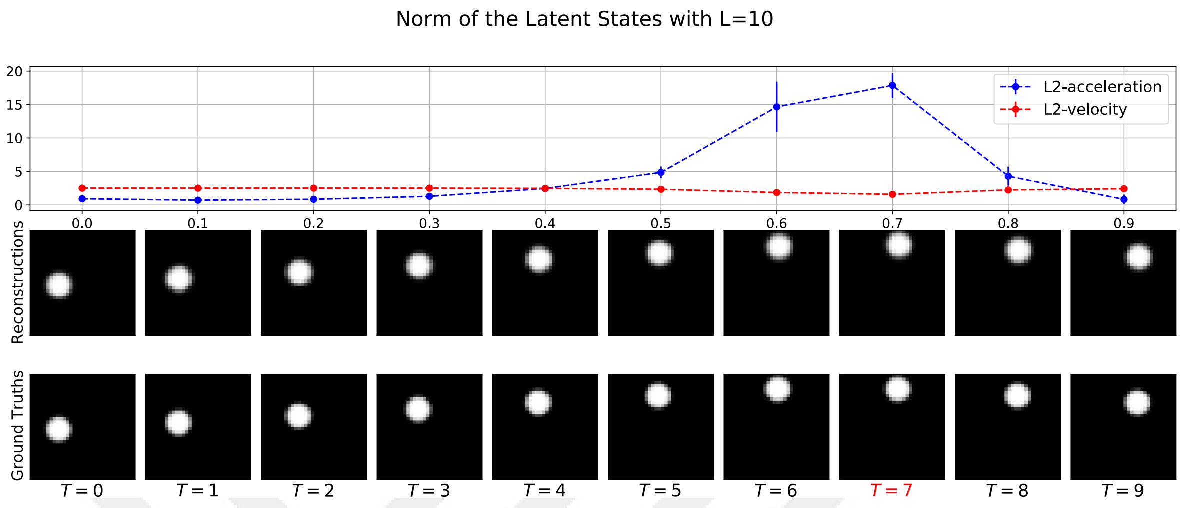
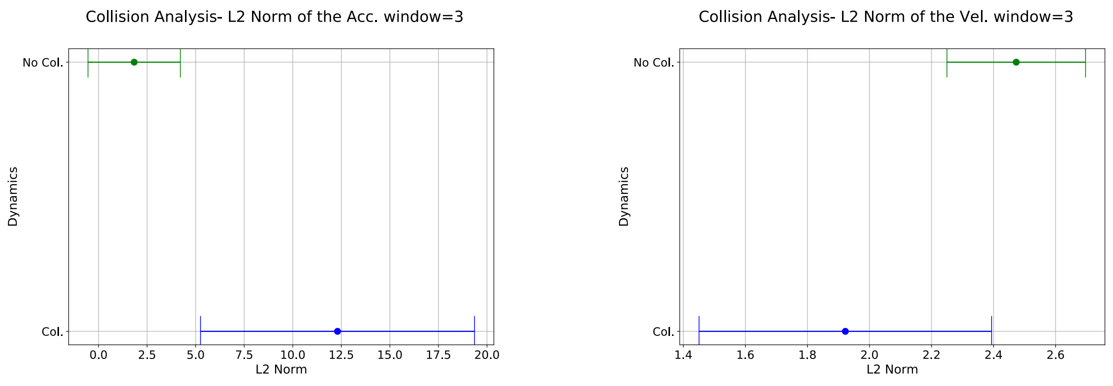


Figure 5.11. Example test case reconstructed by the ODE2VAE model with $a = 3$ for the bouncing balls with $n = 1$. From top to bottom: mean and standard deviation values of the latent norms; mean field prediction by the model; ground truth frames.



a) L2-norm of the Acceleration Latent

b) L2-norm of the Velocity Latent

Figure 5.12. Mean and standard deviation values for the L2-norm of the latent acceleration and velocity of the model for the bouncing balls dataset with $n = 1$. The collision times are expanded with the window size of 3.

Table 5.2. Performance metrics of the selected model on the bouncing balls dataset with $n = 2$. For the MSE and NLL values, the lower is better. For the PSNR scores, the higher is better. Each metric is computed by using 10 samples per test case.

Model \ Metric	MSE	PSNR	NLL
ODE2VAE, $a = 5$	0.0100 ± 0.0123	22.2161 ± 4.3479	96.4297

The baseline model with the latent dimensionality $a = 5$ captures the dynamics of the motion of the two bouncing balls after it is trained for 500 epochs. Increasing the latent dimensionality to $a = 6$ and $a = 9$ has not increased the model performance and has caused convergence issues. In Table 5.2, we report the metrics for the model with $a = 5$. In Figures 5.13 and 5.14, we plot MSE and PSNR values for the test cases over the time steps. The model is able to capture physically meaningful latent representations to a limited extent. We present an example case in Figure 5.15. When the ball hits the wall, there is a spike in the norm of the latent acceleration. However, the acceleration field does not return to its initial magnitude after the collision. Although there is a slight increase in the standard deviation of the norm of the acceleration field during the collision, the uncertainty over the latent acceleration field is persistent over all the time points. Compared to the single bouncing ball case, the uncertainty of the BNN's output is less informative about the dynamics. This may stem from the increased complexity of the motion as the number of balls is doubled. Moreover, the norm of the latent velocity is not preserved during the motion. Figure 5.16a shows that the model generates an acceleration field with a greater norm during the collisions, which is physically meaningful. However, the model is unable to generate a zero acceleration field at the time steps without collision. Figure 5.16b shows that the model cannot preserve the norm of the latent velocity across different test cases, which are supposed to have the same total kinetic energy in the real dynamics.

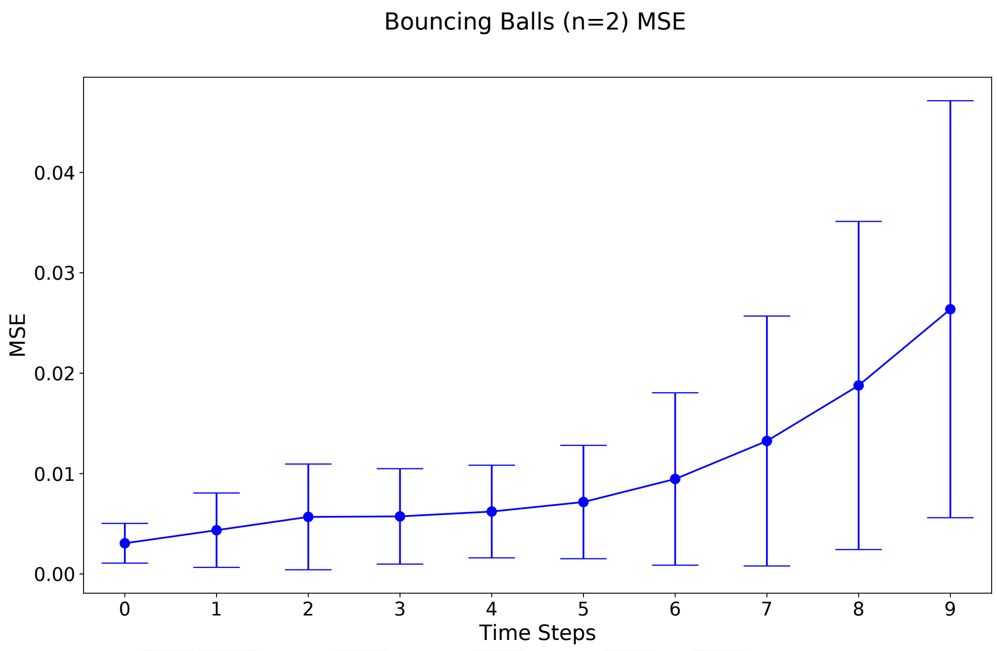


Figure 5.13. MSE values for the bouncing balls dataset with $n = 2$.

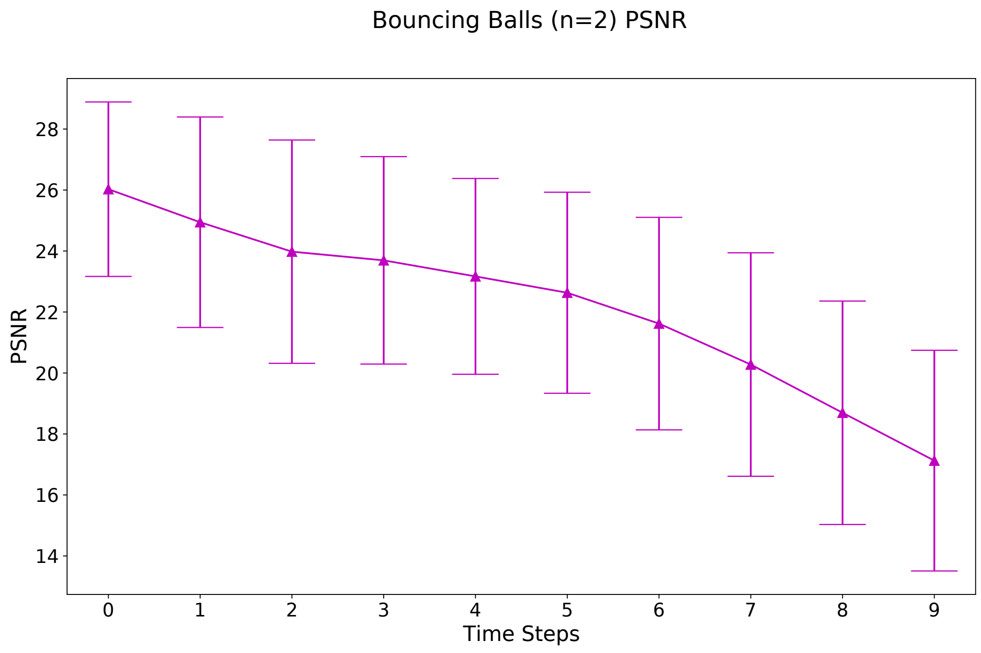


Figure 5.14. PSNR values for the bouncing balls dataset with $n = 2$.

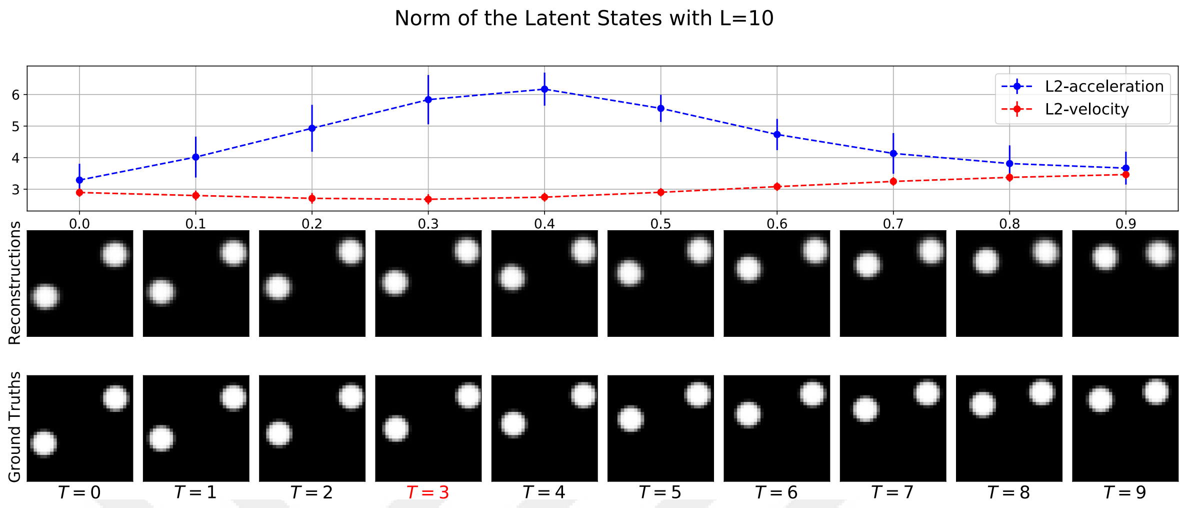
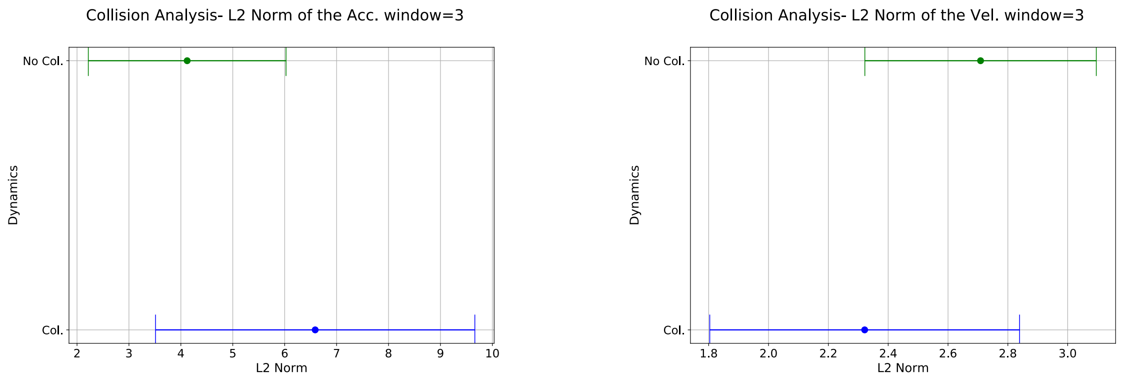


Figure 5.15. Example test case reconstructed by the ODE2VAE model with $a = 5$ for the bouncing balls with $n = 2$. From top to bottom: mean and standard deviation values of the latent norms; mean field prediction by the model; ground truth frames.



a) L2-norm of the Acceleration Latent b) L2-norm of the Velocity Latent

Figure 5.16. Mean and standard deviation values for the L2-norm of the latent acceleration and velocity of the model for the bouncing balls dataset with $n = 2$. The collision times are expanded with the window size of 3.

Table 5.3. Performance metrics of the selected model on the bounding balls dataset with $n = 3$. For the MSE and NLL values, the lower is better. For the PSNR scores, the higher is better. Each metric is computed by using 10 samples per test case.

Metric / Model	MSE	PSNR	NLL
ODE2VAE, $a = 8$	0.0196 ± 0.0174	18.4438 ± 3.4380	154.9332

We find out that the ODE2VAE baseline model could not capture the motion of three bouncing balls with $a = 7$. The baseline model with the latent dimensionality $a = 8$ captures the dynamics of the motion of the three bouncing balls after it is trained for 1000 epochs. Increasing the latent dimensionality to $a = 9$ or $a = 12$ has not increased the model performance. In Table 5.3, we report the metrics for the model with $a = 8$. In Figures 5.17 and 5.18, we plot MSE and PSNR values for the test cases over the time steps. As expected there is an increasing trend in MSE scores and decreasing trend in PSNR scores. We present an example case in Figure 5.19. A careful observation of the reconstructions for the time steps three and four indicates that the model fails to preserve a rigid ball shape. The last frame of the reconstructions also shows that the model lags behind and misses the ball to wall collision. We observe that the magnitude of the acceleration field increases during the collision. However, it starts increasing prior to the collision moments. The uncertainty over the acceleration field's magnitude increases as the model extrapolates into the future. There may be two reasons behind this increased uncertainty, which are the model's reduced predictive performance during the extrapolations and the non-linear motion during the collision. Figure 5.20a shows that the model still generates an acceleration field with a higher magnitude for the time steps with collision. We observe that as the real dynamics get more complex, the model tends to create acceleration fields with closer magnitudes for the collision and no collision moments. Similar to the previous motions, the model is unable to preserve the norm of latent velocity across different test cases (see Figure 5.20b).

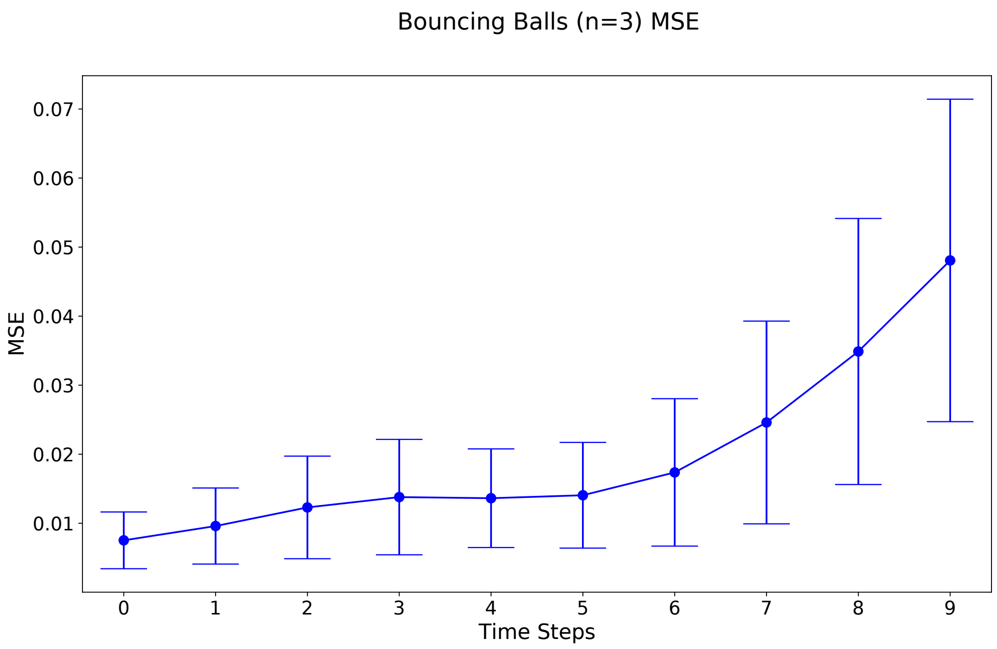


Figure 5.17. MSE values for the bouncing balls dataset with $n = 3$.

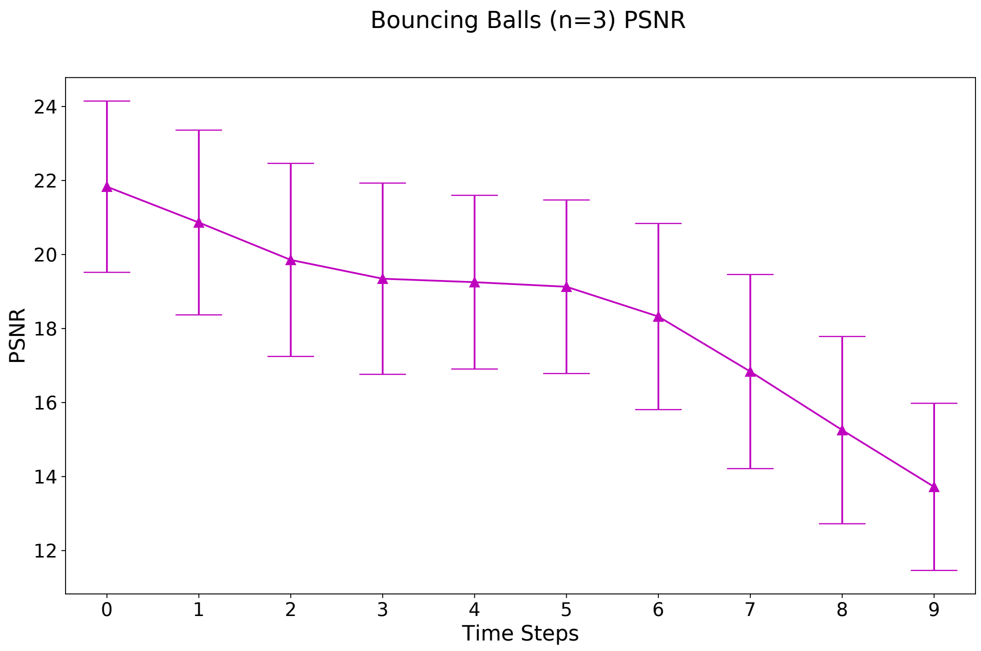


Figure 5.18. PSNR values for the bouncing balls dataset with $n = 3$.

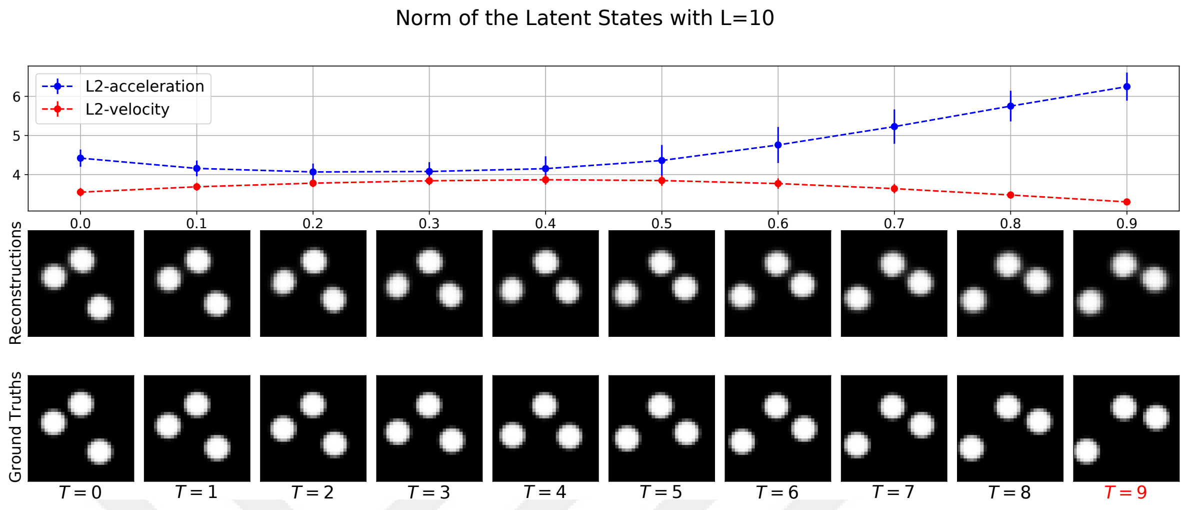
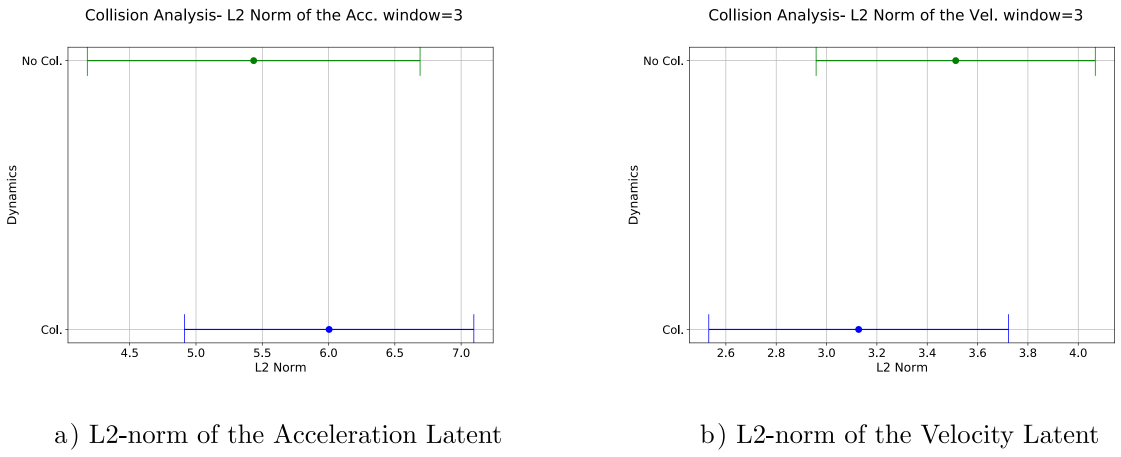


Figure 5.19. Example test case reconstructed by the ODE2VAE model with $a = 8$ for the bouncing balls with $n = 3$. From top to bottom: mean and standard deviation values of the latent norms; mean field prediction by the model; ground truth frames.



a) L2-norm of the Acceleration Latent

b) L2-norm of the Velocity Latent

Figure 5.20. Mean and standard deviation values for the L2-norm of the latent acceleration and velocity of the model for the bouncing balls dataset with $n = 3$. The collision times are expanded with the window size of 3.

Table 5.4. Performance metrics of the selected model on the simple pendulum dataset. Each metric is computed by using 10 samples per test case.

Model \ Metric	MSE	PSNR	NLL
ODE2VAE, $a = 2$	0.0007 ± 0.0006	33.6325 ± 4.5241	26.5609

We train the ODE2VAE model with $a = 2$ and $a = 6$ on the simple pendulum dataset for 300 epochs. Both models have captured the pendulum motion; there is not a considerable improvement between the two models. Since the true pendulum motion can be modeled with two dimensions, the only way for the model to capture the motion with two latent dimensions is by using the decoder to capture the radius of the ball. In Table 5.4, we only report the model's performance with $a = 2$. In Figures 5.21 and 5.22, we plot MSE and PSNR values for the test cases over the time steps. Since the simple pendulum motion is a periodic motion, it may be easier to capture compared to the other motion types. The model is able to capture physically meaningful latent representations. We provide an example case in Figure 5.23 in which the highlighted indices denotes the direction change and the vertical lines on the images denote the equilibrium axis of the pendulum. The model generates an increased norm of the latent acceleration when the object reaches its highest point during its motion and the magnitude of the acceleration is minimized when the object passes through the equilibrium point. Another observation is that the BNN has the decreased uncertainty over the magnitude of the acceleration field when the object passes through the equilibrium point. Additionally, the norm of the latent velocity reaches its peak when the ball passes the equilibrium position and decreases when the object reaches its highest points, which are the highlighted time points. The latent dynamics of the model behave similarly to the true dynamics of the simple pendulum. In Figure 5.24, we provide the statistics for the norm of the latent acceleration and velocity over the test cases with a breakdown for the moments with and without direction change. Figure 5.24a shows that the model generates an acceleration field with a greater magnitude when the object reaches its highest point.

Figure 5.24b shows that the latent velocities have a reduced norm at the moments of direction change. Although the model cannot generate velocity latents with magnitude zero during direction change, it captures a physically plausible decreasing trend.

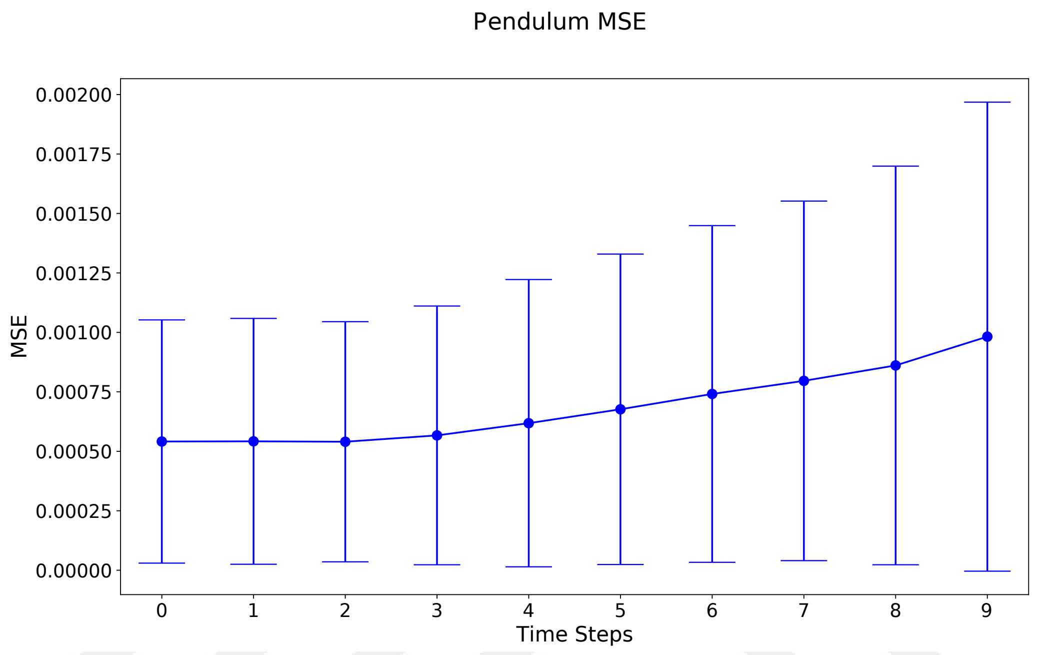


Figure 5.21. MSE values for the simple pendulum test set.

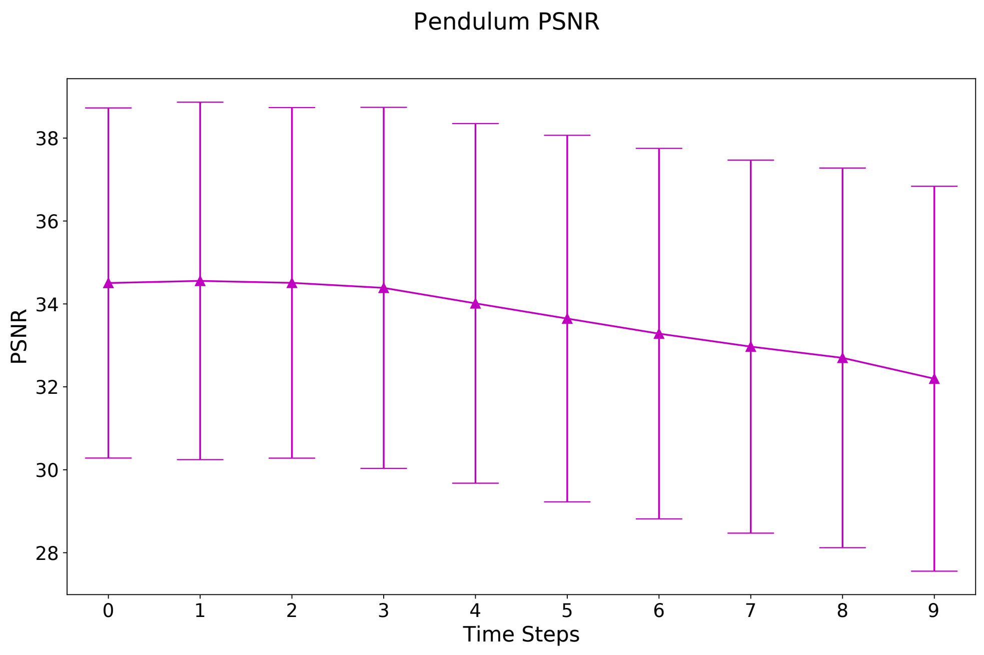


Figure 5.22. PSNR values for the simple pendulum test set.

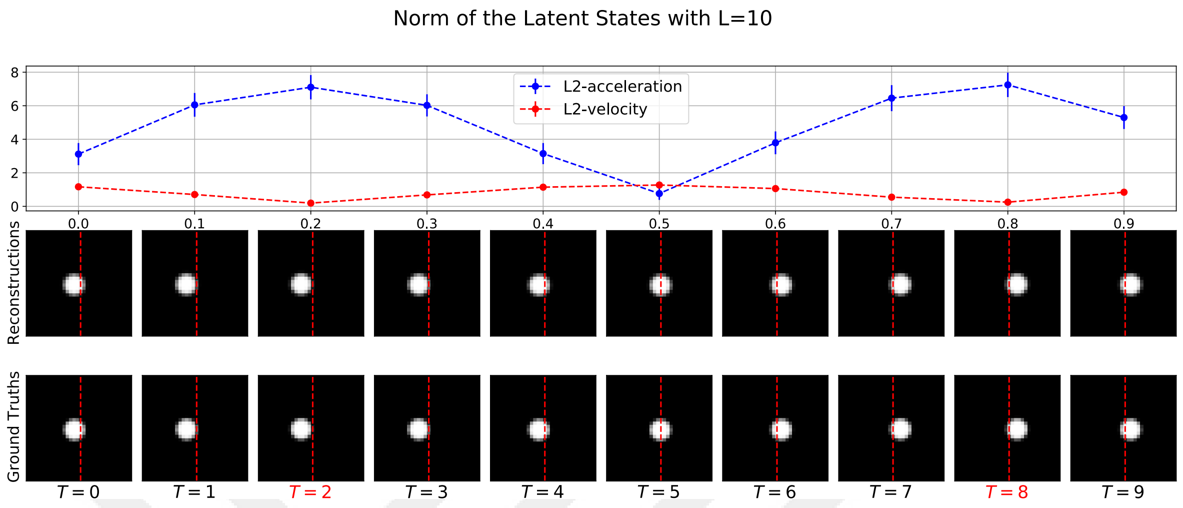
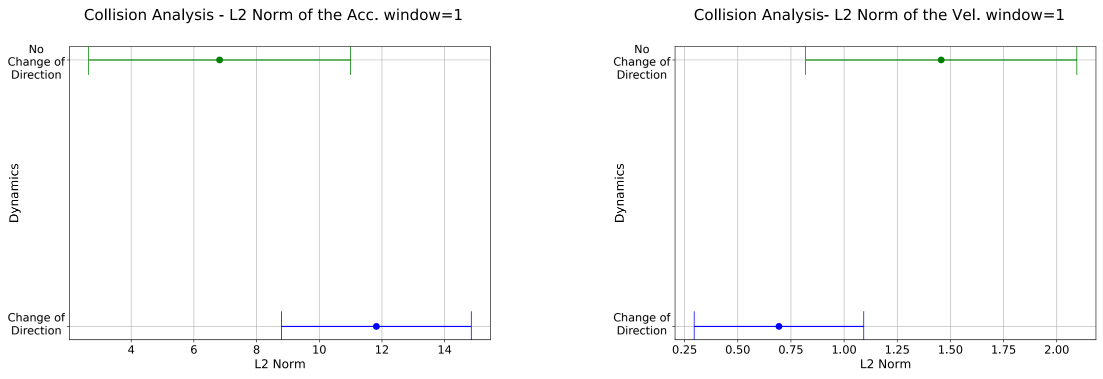


Figure 5.23. Example test case reconstructed by the ODE2VAE model with $a = 2$ for the simple pendulum dataset. From top to bottom: mean and standard deviation values of the latent norms; mean field prediction by the model; ground truth frames.



a) L2-norm of the Acceleration Latent

b) L2-norm of the Velocity Latent

Figure 5.24. Mean and standard deviation values for the L2-norm of the latent acceleration and velocity the pendulum dataset. The figures display the values at the moment of direction change and other time steps.

Table 5.5. Performance metrics of the selected model on the projectile motion dataset. Each metric is computed by using 10 samples per test case.

Metric / Model	MSE	PSNR	NLL
ODE2VAE, $a = 9$	0.0016 ± 0.0016	30.5322 ± 5.1913	27.8267

We find out that the ODE2VAE baseline model could not capture the projectile motion dynamics with $a = 2, 3, 5, 7$ after it is trained for 300 epochs and converged. Therefore, we increased the number of latent units to 9. Although the model has increased predictive performance with $a = 9$, it is not fully able to capture latent dynamics that resemble the real projectile motion. In Figures 5.25 and 5.26, we plot MSE and PSNR values for the test cases over the time steps. After time step four, there is an interval in which the ball hits the ground for each test case. Since the ball spends 0.1 second during its collision and the frames are separated for 0.1 second, the motion of the ball becomes stationary. The drop in the MSE values and the peak in the PSNR values may be because of the network's tendency to overfit when the ball becomes stationary. We present an example case in Figure 5.27. One of the challenges about the projectile motion is that there is a constant gravitational acceleration to learn. The results show that the model is not able to generate a constant acceleration field. On the other hand, there is a slight increment in the total kinetic energy when the ball is closer to the ground. In Figure 5.28a and 5.28b, we present the norm of the acceleration and velocity latents at the time steps with and without collision. Although the true acceleration field is constant (except for the collision moments) due to the constant free-fall acceleration, the model is not able to learn a fixed acceleration field among the different test cases (see Figure 5.28a). We omit to comment on the norm of the latent velocity since the ground truth velocities are not the same in the test set. It is only reported for the sake of completeness.

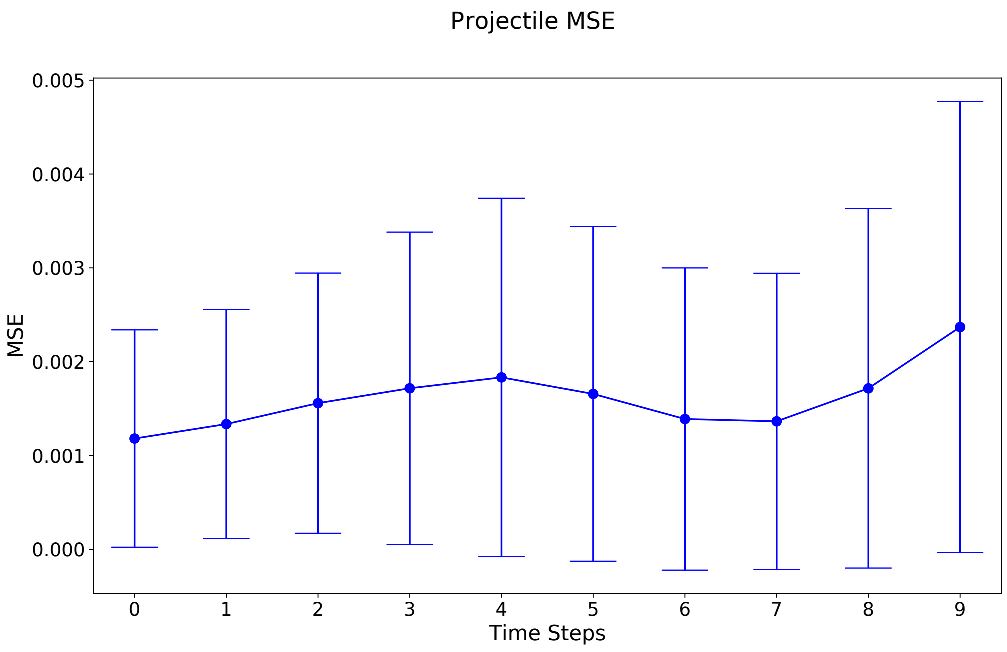


Figure 5.25. MSE values for the projectile motion test set.

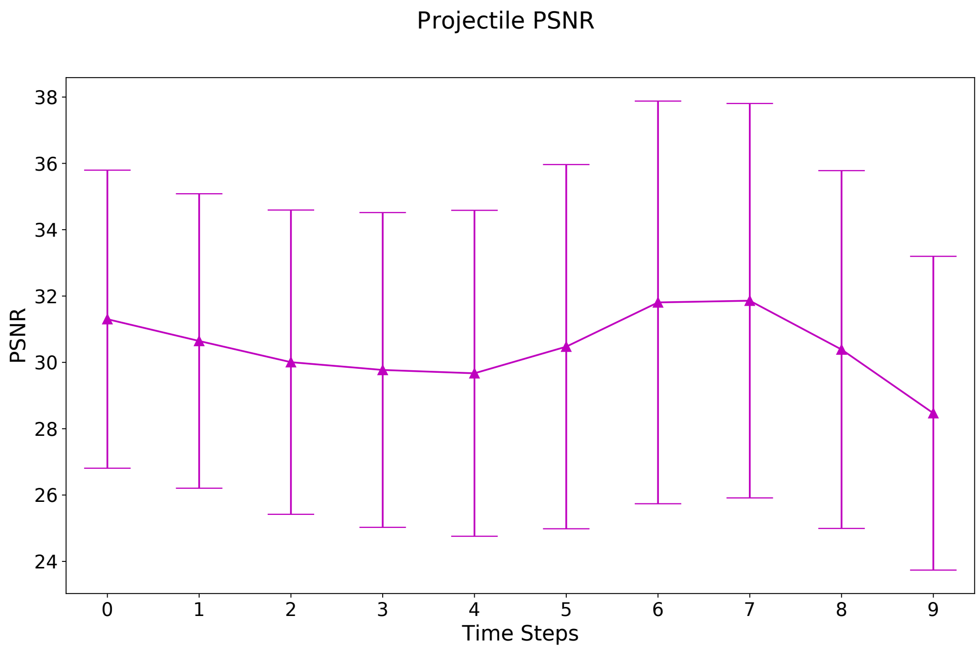


Figure 5.26. PSNR values for the projectile motion test set.

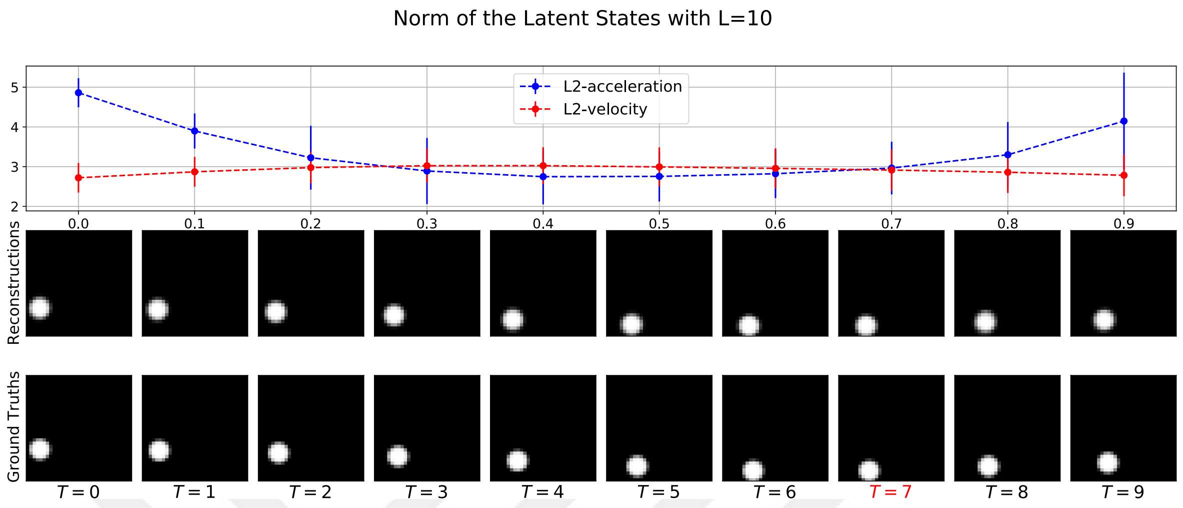
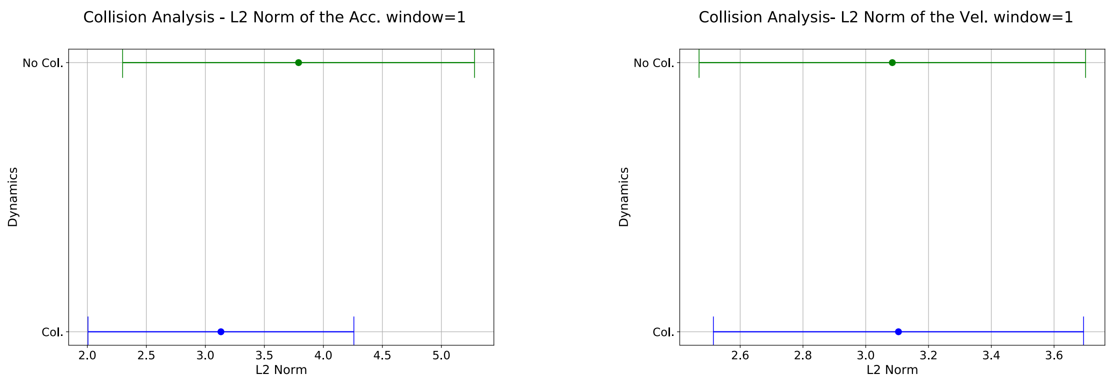


Figure 5.27. Example test case reconstructed by the ODE2VAE model with $a = 9$ for the projectile motion dataset. From top to bottom: mean and standard deviation values of the latent norms; mean field prediction by the model; ground truth frames.



a) L2-norm of the Acceleration Latent

b) L2-norm of the Velocity Latent

Figure 5.28. Mean and standard deviation values for the L2-norm of the latent acceleration and velocity for the projectile motion dataset. The figures display the values at the moment of collisions and other time steps.

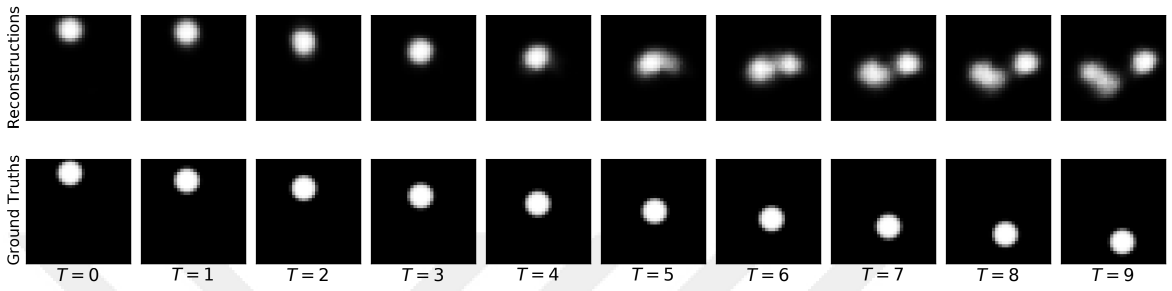
5.3.3. Learning Varying Static Features with the ODE2VAE Model

In this section, we challenge the ODE2VAE model by using three novel datasets: mixed bouncing balls with $n = 1, 2, 3$, bouncing balls with different radii, and bouncing dSprites. These datasets have varying static features among their cases. Therefore, they can be used to check if the ODE2VAE model can learn static features instead of memorizing them through the decoder. Although the proposed models β-ODE2VAE and ODE2VAE-c may have a proper formulation for learning static features, they increase the model complexity and cause convergence issues during training. Therefore, we have limited the results in this section to the performance of the baseline model. We remind that the baseline model's performance on the datasets with fixed static features is reported in the previous section.

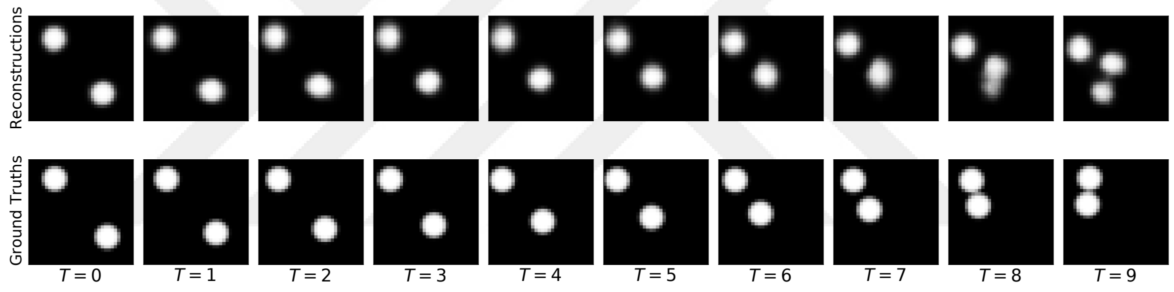
Table 5.6. Performance metrics of the selected model on the mixed bouncing balls dataset. Each metric is computed by using 10 samples per test case.

Metric / Model	MSE	PSNR	NLL
ODE2VAE, $a = 15$	0.0346 ± 0.0298	16.4827 ± 4.4092	190.0550
ODE2VAE, $a = 20$	0.0323 ± 0.0262	16.5108 ± 3.9572	176.5782

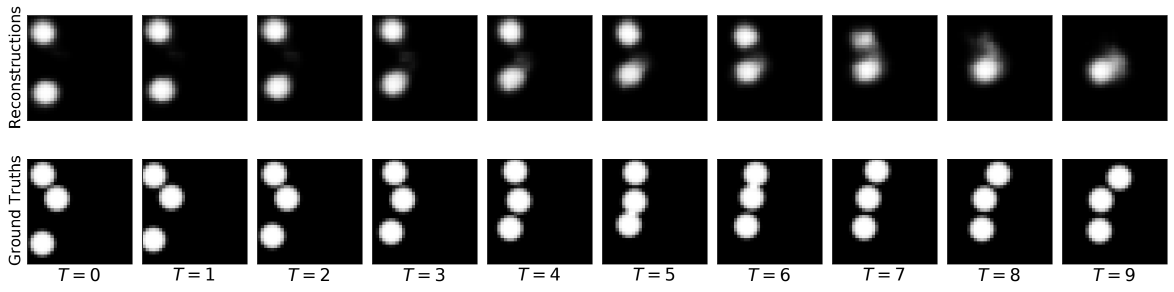
We train the baseline model with $a = 15$ and 20 for 1000 epochs on the mixed bouncing balls dataset. The metrics are reported in Table 5.6. The model with $a = 20$ captures the motion better than the model with $a = 15$. In Figure 5.29, we present example mean field reconstructions of the baseline model with $a = 20$ with the ground truths. Although the model learns the independent bouncing ball cases, it is clearly shown that the model cannot capture the varying number of balls in the test cases. In Figure 5.29, it can be seen that the model cannot preserve the number of balls as it extrapolates into the future. Some possible reasons behind the model's inability may be the convergence issues and the model's dynamical inductive bias which may hinder preserving static features over the time steps.



a) Reconstruction of the ODE2VAE model with $a = 20$ for the mixed bouncing balls dataset.



b) Reconstruction of the ODE2VAE model with $a = 20$ for the mixed bouncing balls dataset.



c) Reconstruction of the ODE2VAE model with $a = 20$ for the mixed bouncing balls dataset.

Figure 5.29. Figure for mean field reconstruction of the baseline model with $a = 20$ and ground truth frames for the mixed bouncing balls dataset.

Table 5.7. Performance metrics of the selected models on the bouncing balls with multiple radii dataset. Each metric is computed by using 10 samples per test case.

Metric / Model	MSE	PSNR	NLL
ODE2VAE, $a = 5$	0.0199 ± 0.0137	18.7689 ± 4.7620	136.0506
ODE2VAE, $a = 10$	0.0175 ± 0.0137	18.8410 ± 3.4849	97.0111
ODE2VAE, $a = 15$	0.0172 ± 0.0144	19.8084 ± 5.1230	103.7041

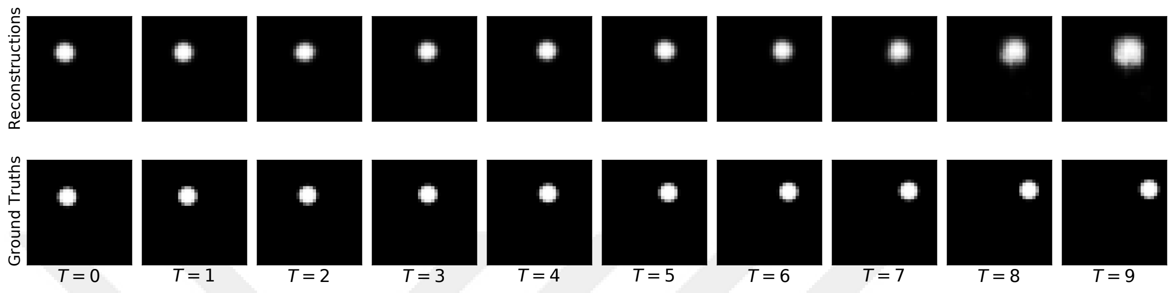
Secondly, we train the ODE2VAE model with $a = 5, 10$, and 15 for 1000 epochs on the bouncing balls dataset with multiple radii. We report the results in Table 5.7. In Figure 5.30, we present mean field reconstructions for each model variant. The models cannot capture the varying radius of the balls. In Figure 5.30a, we observe that the model cannot preserve the shape of the ball, which may stem from the low latent dimensionality $a = 5$. In Figures 5.30b and 5.30c, we observe that the models collapse to a smaller radius. They learn the dynamics, but they could not capture the true radius. Also, the model could not preserve the rigid ball shape in Figure 5.30c.

Table 5.8. Performance metrics of the selected model on the bouncing dSprites dataset. Each metric is computed by using 10 samples per test case.

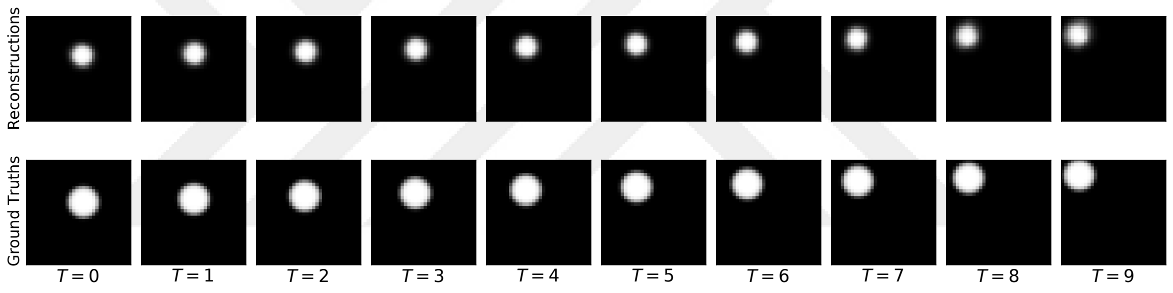
Metric / Model	MSE	PSNR	NLL
ODE2VAE, $a = 15$	0.0384 ± 0.0288	15.1704 ± 2.9083	142.2924

Lastly, we train the baseline ODE2VAE model with $a = 5, 10$, and 15 for 1000 epochs on the bouncing dSprites dataset. In Table 5.8, we only report the results for the $a = 15$ since the other models did not capture the dataset's motion. As shown in Figure 5.31a, the model with $a = 15$ could capture the motion and the shape to a limited extend. Even though the model generates the object with a square like shape in the beginning, it could not preserve the size and shape.

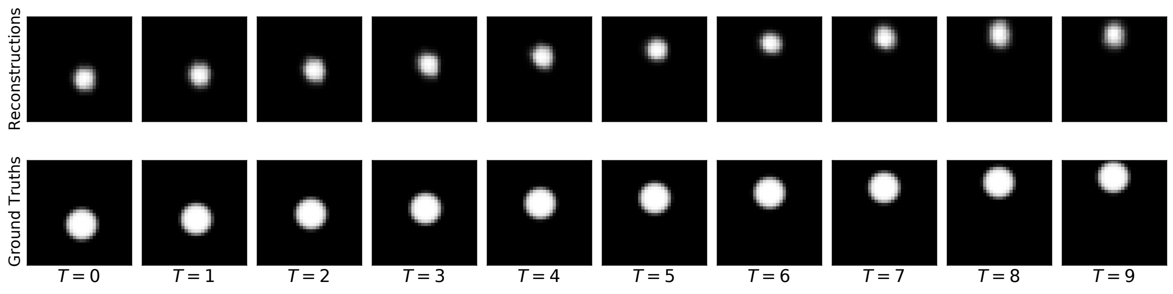
Given that the model could learn the bouncing balls motion, learning different shapes seems to hinder the model's ability to learn dynamic features. It also fails to generate motion after the sixth time step. Moreover, the model tends to collapse to a single shape as shown in Figure 5.31b.



a) Mean field reconstruction of the ODE2VAE model with $a = 5$ for the multiple radii dataset.

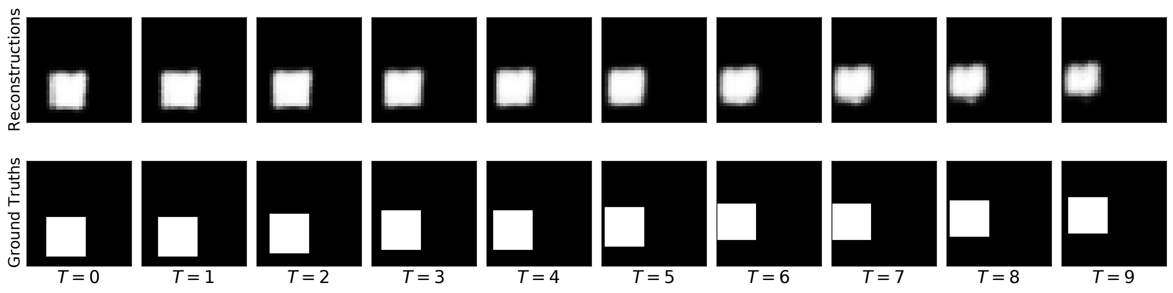


b) Mean field reconstruction of the ODE2VAE model with $a = 10$ for the multiple radii dataset.

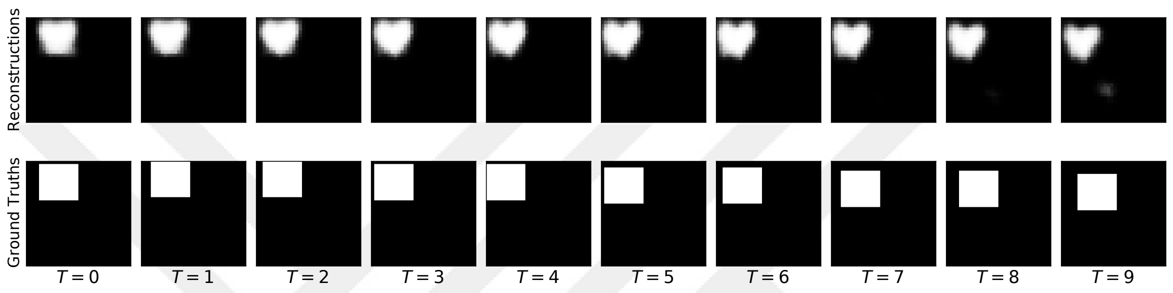


c) Mean field reconstruction of the ODE2VAE model with $a = 15$ for the multiple radii dataset.

Figure 5.30. Figure for mean field reconstruction of the baseline models with $a = 5$, 10, 15 and ground truth frames for the multiple radii dataset.



a) Mean field reconstruction of the ODE2VAE model with $a = 15$ for the bouncing dSprites dataset.



b) Mean field reconstruction of the ODE2VAE model with $a = 15$ for the bouncing dSprites dataset.

Figure 5.31. Figure for mean field reconstruction of the baseline model with $a = 15$ and ground truth frames for the bouncing dSprites dataset.

6. CONCLUSIONS AND FUTURE WORK

In this thesis, we work on a generative model, ODE2VAE [5], which uses second order latent ODEs and Bayesian neural networks to learn arbitrary dynamics of the sequences in an unsupervised setting. The model has an inductive bias that is imposed by the coupled latent ODEs, which learn arbitrary latent dynamics. However, the effects of the inductive bias are not investigated in the original work [5]. We challenge the model with different datasets and attempt to show the effects of the inductive bias of the ODE2VAE model over its latent representations. We choose to work on well-known physical motion datasets: bouncing balls, simple pendulum, and projectile motion datasets. We investigate if the learned latent representations behave according to the real motion dynamics. Since the latent representations are high dimensional, we use the norm of the latent representations during our analysis. We are able to show that the ODE2VAE model can learn physically plausible dynamic latent representations for the single bouncing ball and simple pendulum datasets. Although the model generates successful predictions for the projectile motion dataset, its latent representations lack physical intuition. Our results also show that the uncertainty over the magnitude of the acceleration field increases during rare events and non-linear motions such as collisions.

In the second part, we challenge the baseline model to learn different static features in addition to the dynamic features, which was not investigated previously [5]. In the experiments, we utilize three novel datasets: mixed bouncing balls, bouncing balls with different radii, and bouncing dSprites. These datasets differ from the previous ones since they have varying static features among their cases. We re-formulate the model for a flexible regularization (β-ODE2VAE) and attempt to learn static features efficiently through regularization. In addition, we attempt to learn static features by extending the baseline model's architecture (ODE2VAE-c). Given the fact that the baseline model has a complex architecture and unstable training, we have not been able to optimize the model variants, which are more complex than the baseline model.

Therefore, we only investigate the performance of the baseline model trained over the sequences with multiple static features. Our findings show a possible pitfall of the baseline model. Although the model could learn arbitrary dynamics of the bouncing balls motion with fixed static features, it fails to learn the same motion with varying static features. This observation suggests that the model's dynamical inductive bias is not suitable for learning different static features that should be preserved over the time steps.

Our future work will include preserving latent physical quantities by parameterizing the latent acceleration field by utilizing arbitrary Lagrangians or Hamiltonians [15, 16]. Simplifying the ODE2VAE model's formulation may be a long-term goal, which may reduce the model's complexity and solve the convergence issues that we faced during optimization. As the learning of static features through regularization and architectural extension complicates the model, more straightforward approaches such as hard-coding zeros to corresponding acceleration and velocity latent units will be included in future work.

REFERENCES

1. Kingma, D. P. and M. Welling, "Auto-Encoding Variational Bayes", *2nd International Conference on Learning Representations (ICLR)*, 2014.

2. Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville and Y. Bengio, "Generative Adversarial Nets", *Advances in Neural Information Processing Systems*, Vol. 27, pp. 2672–2680, Curran Associates, Inc., 2014.

3. Ruthotto, L. and E. Haber, "An Introduction to Deep Generative Modeling", *CoRR*, Vol. abs/2103.05180, 2021.

4. Karras, T., T. Aila, S. Laine and J. Lehtinen, "Progressive Growing of GANs for Improved Quality, Stability, and Variation", *6th International Conference on Learning Representations (ICLR)*, OpenReview.net, 2018.

5. Yildiz, C., M. Heinonen and H. Lähdesmäki, "ODE2VAE: Deep Generative Second Order ODEs with Bayesian Neural Networks", *Advances in Neural Information Processing Systems*, Vol. 32, pp. 13412–13421, Curran Associates, Inc., 2019.

6. Rubanova, Y., T. Q. Chen and D. Duvenaud, "Latent Ordinary Differential Equations for Irregularly-Sampled Time Series.", *Advances in Neural Information Processing Systems*, Vol. 32, pp. 5321–5331, Curran Associates, Inc., 2019.

7. Fortuin, V., D. Baranchuk, G. Rätsch and S. Mandt, "GP-VAE: Deep Probabilistic Time Series Imputation", *The 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, Vol. 108 of *Proceedings of Machine Learning Research*, pp. 1651–1661, PMLR, 2020.

8. van den Oord, A., O. Vinyals and K. Kavukcuoglu, "Neural Discrete Representation Learning", *Advances in Neural Information Processing Systems*, Vol. 30, pp.

6306–6315, Curran Associates, Inc., 2017.

9. Fortuin, V., M. Hüser, F. Locatello, H. Strathmann and G. Rätsch, "SOM-VAE: Interpretable Discrete Representation Learning on Time Series", *7th International Conference on Learning Representations (ICLR)*, OpenReview.net, 2019.

10. An, J. and S. Cho, "Variational Autoencoder Based Anomaly Detection Using Reconstruction Probability", *Special Lecture on IE*, Vol. 2, No. 1, 2015.

11. Dilokthanakul, N., P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran and M. Shanahan, "Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders", *CoRR*, Vol. abs/1611.02648, 2016.

12. Schönfeld, E., S. Ebrahimi, S. Sinha, T. Darrell and Z. Akata, "Generalized Zero- and Few-Shot Learning via Aligned Variational Autoencoders", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8247–8255, Computer Vision Foundation / IEEE, 2019.

13. Goyal, A. and Y. Bengio, "Inductive Biases for Deep Learning of Higher-Level Cognition", *CoRR*, Vol. abs/2011.15091, 2020.

14. Battaglia, P. W., J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li and R. Pascanu, "Relational Inductive Biases, Deep Learning, and Graph Networks", *CoRR*, Vol. abs/1806.01261, 2018.

15. Toth, P., D. J. Rezende, A. Jaegle, S. Racanière, A. Botev and I. Higgins, "Hamiltonian Generative Networks", *8th International Conference on Learning Representations (ICLR)*, OpenReview.net, 2020.

16. Cranmer, M. D., S. Greydanus, S. Hoyer, P. W. Battaglia, D. N. Spergel and S. Ho,

"Lagrangian Neural Networks", *CoRR*, Vol. abs/2003.04630, 2020.

17. Chen, R. T. Q., Y. Rubanova, J. Bettencourt and D. Duvenaud, "Neural Ordinary Differential Equations", *Advances in Neural Information Processing Systems*, Vol. 31, pp. 6572–6583, Curran Associates, Inc., 2018.

18. Higgins, I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed and A. Lerchner, "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework", *5th International Conference on Learning Representations (ICLR)*, OpenReview.net, 2017.

19. Bengio, Y., "Deep Learning of Representations: Looking Forward", *Statistical Language and Speech Processing (SLSP) - First International Conference*, Vol. 7978 of *Lecture Notes in Computer Science*, pp. 1–37, Springer, 2013.

20. Jospin, L. V., W. L. Buntine, F. Boussaïd, H. Laga and M. Bennamoun, "Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users", *CoRR*, Vol. abs/2007.06823, 2020.

21. Lombardo, S., J. Han, C. Schroers and S. Mandt, "Deep Generative Video Compression", *Advances in Neural Information Processing Systems*, Vol. 32, pp. 9283–9294, Curran Associates, Inc., 2019.

22. LeCun, Y., Y. Bengio and G. Hinton, "Deep Learning", *Nature*, Vol. 521, No. 7553, pp. 436–444, 2015.

23. Rumelhart, D. E., G. E. Hinton and R. J. Williams, "Learning Representations by Back-Propagating Errors", *Nature*, Vol. 323, No. 6088, pp. 533–536, 1986.

24. Duchi, J., E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, Vol. 12, pp. 2121–2159, 2011.

25. Kingma, D. P. and J. Ba, “Adam: A Method for Stochastic Optimization”, *3rd International Conference on Learning Representations (ICLR)*, 2015.

26. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016.

27. Fukushima, K., “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”, *Biological Cybernetics*, Vol. 36, No. 4, 1980.

28. Cybenko, G., “Approximation by Superpositions of a Sigmoidal Function”, *Math. Control. Signals Syst.*, Vol. 2, No. 4, pp. 303–314, 1989.

29. Hornik, K., “Approximation Capabilities of Multilayer Feedforward Networks”, *Neural Networks*, Vol. 4, No. 2, pp. 251–257, 1991.

30. Lecun, Y., L. Bottou, Y. Bengio and P. Haffner, “Gradient-Based Learning Applied to Document Recognition”, *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

31. Krizhevsky, A., I. Sutskever and G. E. Hinton, “ImageNet Classification With Deep Convolutional Neural Networks”, *Advances in Neural Information Processing Systems*, Vol. 25, pp. 1106–1114, Curran Associates, Inc., 2012.

32. Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge”, *International Journal of Computer Vision (IJCV)*, Vol. 115, No. 3, pp. 211–252, 2015.

33. Chung, J., K. Kastner, L. Dinh, K. Goel, A. C. Courville and Y. Bengio, “A Recurrent Latent Variable Model for Sequential Data.”, *Advances in Neural Information Processing Systems*, Vol. 28, pp. 2980–2988, Curran Associates, Inc., 2015.

34. Pu, Y., Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens and L. Carin, "Variational Autoencoder for Deep Learning of Images, Labels and Captions.", *Advances in Neural Information Processing Systems*, Vol. 29, pp. 2352–2360, Curran Associates, Inc., 2016.

35. Jordan, M. I., Z. Ghahramani, T. S. Jaakkola and L. K. Saul, "Introduction to Variational Methods for Graphical Models", *Machine Learning*, Vol. 37, No. 2, 1999.

36. Blei, D. M., A. Kucukelbir and J. D. McAuliffe, "Variational Inference: A Review for Statisticians", *Journal of the American Statistical Association*, Vol. 112, No. 518, p. 859–877, 2017.

37. Altosaar, J., *Probabilistic Modeling of Structure in Science: Statistical Physics to Recommender Systems*, Ph.D. Thesis, Princeton University, 2020.

38. Murphy, K. P., *Machine Learning: A Probabilistic Perspective*, The MIT Press, Cambridge, MA, 2012.

39. Kingma, D. P. and M. Welling, "An Introduction to Variational Autoencoders", *Foundations and Trends in Machine Learning*, Vol. 12, No. 4, pp. 307–392, 2019.

40. Gershman, S. and N. D. Goodman, "Amortized Inference in Probabilistic Reasoning", *Proceedings of the 36th Annual Meeting of the Cognitive Science*, pp. 517–522, 2014.

41. Kingma, D. P., *Variational Inference & Deep Learning: A New Synthesis*, Ph.D. Thesis, University of Amsterdam, 2017.

42. Rezende, D. J., S. Mohamed and D. Wierstra, "Stochastic Backpropagation and Approximate Inference in Deep Generative Models", *31st International Conference on Machine Learning*, Vol. 4, pp. 3057–3070, International Machine Learning Society (IMLS), 2014.

43. Altosaar, J., *Tutorial - What is a Variational Autoencoder?*, 2016, `https://doi.org/10.5281/zenodo.4462916`, accessed in March 2021.

44. Burgess, C. P., I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins and A. Lerchner, "Understanding Disentangling in beta-VAE", *CoRR*, Vol. abs/1804.03599, 2018.

45. Kingma, D. P., S. Mohamed, D. J. Rezende and M. Welling, "Semi-supervised Learning with Deep Generative Models", *Advances in Neural Information Processing Systems*, Vol. 27, pp. 3581–3589, Curran Associates, Inc., 2014.

46. Sohn, K., H. Lee and X. Yan, "Learning Structured Output Representation using Deep Conditional Generative Models", *Advances in Neural Information Processing Systems*, Vol. 28, pp. 3483–3491, Curran Associates, Inc., 2015.

47. Walker, J., C. Doersch, A. Gupta and M. Hebert, "An Uncertain Future: Forecasting from Static Images using Variational Autoencoders", *CoRR*, Vol. abs/1606.07873, 2016.

48. Mehrasa, N., A. A. Jyothi, T. Durand, J. He, L. Sigal and G. Mori, "A Variational Auto-Encoder Model for Stochastic Point Processes", *CoRR*, Vol. abs/1904.03273, 2019.

49. Ainsworth, S. K., N. J. Foti and E. B. Fox, "Disentangled VAE Representations for Multi-Aspect and Missing Data", *CoRR*, Vol. abs/1806.09060, 2018.

50. Xian, Y., S. Sharma, B. Schiele and Z. Akata, "f-VAEGAN-D2: A Feature Generating Framework for Any-Shot", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10275–10284, Computer Vision Foundation / IEEE, 2019.

51. Bengio, Y., A. C. Courville and P. Vincent, "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives", *CoRR*, Vol. abs/1206.5538,

2012.

52. van Steenkiste, S., F. Locatello, J. Schmidhuber and O. Bachem, “Are Disentangled Representations Helpful for Abstract Visual Reasoning?”, *CoRR*, Vol. abs/1905.12506, 2019.

53. Kim, H. and A. Mnih, “Disentangling by Factorising”, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Vol. 80, pp. 2654–2663, PMLR, 2018.

54. Chen, R. T. Q., X. Li, R. B. Grosse and D. K. Duvenaud, “Isolating Sources of Disentanglement in Variational Autoencoders”, *Advances in Neural Information Processing Systems*, Vol. 31, pp. 2615–2625, Curran Associates, Inc., 2018.

55. Chen, X., Y. Duan, R. Houthooft, J. Schulman, I. Sutskever and P. Abbeel, “InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets”, *Advances in Neural Information Processing Systems*, Vol. 29, pp. 2172–2180, Curran Associates, Inc., 2016.

56. Denton, E. L. and v. Birodkar, “Unsupervised Learning of Disentangled Representations from Video”, *Advances in Neural Information Processing Systems*, Vol. 30, pp. 4414–4423, Curran Associates, Inc., 2017.

57. Hsieh, J., B. Liu, D. Huang, F. Li and J. C. Niebles, “Learning to Decompose and Disentangle Representations for Video Prediction”, *Advances in Neural Information Processing Systems*, Vol. 31, pp. 515–524, Curran Associates, Inc., 2018.

58. Zhu, D., M. Munderloh, B. Rosenhahn and J. Stückler, “Learning to Disentangle Latent Physical Factors for Video Prediction”, *German Conference on Pattern Recognition (GCPR)*, 2019.

59. Li, Y. and S. Mandt, “Disentangled Sequential Autoencoder”, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Vol. 80, pp. 5656–

5665, PMLR, 2018.

60. Bengio, Y., P. Simard and P. Frasconi, “Learning Long-Term Dependencies with Gradient Descent Is Difficult”, *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp. 157–166, 1994.

61. Hochreiter, S. and J. Schmidhuber, “Long Short-Term Memory”, *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

62. Haber, E. and L. Ruthotto, “Stable Architectures for Deep Neural Networks.”, *CoRR*, Vol. abs/1705.03341, 2017.

63. Lu, Y., A. Zhong, Q. Li and B. Dong, “Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations”, *CoRR*, Vol. abs/1710.10121, 2017.

64. Khalil, H. K., *Nonlinear Systems; 3rd ed.*, Prentice-Hall, Upper Saddle River, NJ, 2002.

65. Kendall E., A., H. Weimin and S. David, *Numerical Solution of Ordinary Differential Equations*, John Wiley & Sons, Ltd, Hoboken, NJ, 2009.

66. He, K., X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition”, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, IEEE Computer Society, 2016.

67. Chang, B., L. Meng, E. Haber, L. Ruthotto, D. Begert and E. Holtham, “Reversible Architectures for Arbitrarily Deep Residual Neural Networks”, *CoRR*, Vol. abs/1709.03698, 2017.

68. Pontryagin, L. S., V. G. Boltyanskii, R. V. Gamkrelidze and E. F. Mishechenko, *The Mathematical Theory of Optimal Processes*, Interscience, New York, 1962.

69. Rezende, D. J. and S. Mohamed, “Variational Inference with Normalizing Flows”, *32nd International Conference on Machine Learning (ICML)*, Vol. 2, 2015.

70. Grathwohl, W., R. T. Chen, J. Bettencourt, I. Sutskever and D. Duvenaud, “Ffjord: Free-Form Continuous Dynamics for Scalable Reversible Generative Models”, *7th International Conference on Learning Representations (ICLR)*, OpenReview.net, 2019.

71. Massaroli, S., M. Poli, J. Park, A. Yamashita and H. Asama, “Dissecting Neural ODEs”, *Advances in Neural Information Processing Systems*, Vol. 33, pp. 3952–3963, Curran Associates, Inc., 2020.

72. Ott, K., P. Katiyar, P. Hennig and M. Tiemann, “When are Neural ODE Solutions Proper ODEs?”, *CoRR*, Vol. abs/2007.15386, 2020.

73. Ghosh, A., H. S. Behl, E. Dupont, P. H. S. Torr and V. Namboodiri, “STEER : Simple Temporal Regularization For Neural ODEs”, *CoRR*, Vol. abs/2006.10711, 2020.

74. De Brouwer, E., J. Simm, A. Arany and Y. Moreau, “GRU-ODE-Bayes: Continuous Modeling of Sporadically-Observed Time Series”, *Advances in Neural Information Processing Systems*, Vol. 32, pp. 7377–7388, Curran Associates, Inc., 2019.

75. Kidger, P., J. Morrill, J. Foster and T. Lyons, “Neural Controlled Differential Equations for Irregular Time Series”, *Advances in Neural Information Processing Systems*, Vol. 33, pp. 6696–6707, Curran Associates, Inc., 2020.

76. Chen, R. T. Q., B. Amos and M. Nickel, “Learning Neural Event Functions for Ordinary Differential Equations”, *9th International Conference on Learning Representations (ICLR)*, OpenReview.net, 2021.

77. Greydanus, S., M. Dzamba and J. Yosinski, “Hamiltonian Neural Networks”,

CoRR, Vol. abs/1906.01563, 2019.

78. Gwak, D., G. Sim, M. Poli, S. Massaroli, J. Choo and E. Choi, "Neural Ordinary Differential Equations for Intervention Modeling", *CoRR*, Vol. abs/2010.08304, 2020.

79. Øksendal, B., *Stochastic Differential Equations*, pp. 65–84, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

80. Li, X., T. L. Wong, R. T. Q. Chen and D. Duvenaud, "Scalable Gradients for Stochastic Differential Equations", *The 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, Vol. 108 of *Proceedings of Machine Learning Research*, pp. 3870–3882, PMLR, 2020.

81. Dandekar, R., V. Dixit, M. Tarek, A. Garcia-Valadez and C. Rackauckas, "Bayesian Neural Ordinary Differential Equations", *CoRR*, Vol. abs/2012.07244, 2020.

82. Gal, Y., *Uncertainty in Deep Learning*, Ph.D. Thesis, University of Cambridge, 2016.

83. Litjens, G., T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken and C. I. Sánchez, "A Survey on Deep Learning in Medical Image Analysis", *Medical Image Analysis*, Vol. 42, pp. 60–88, 2017.

84. Esteva, A., A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun and J. Dean, "A Guide to Deep Learning in Healthcare", *Nature Medicine*, Vol. 25, No. 1, pp. 24–29, 2019.

85. Sünderhauf, N., O. Brock, W. J. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford and P. Corke, "The Limits and Potentials of Deep Learning for Robotics", *CoRR*, Vol. abs/1804.06557, 2018.

86. Levine, S., P. Pastor, A. Krizhevsky and D. Quillen, “Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection”, *CoRR*, Vol. abs/1603.02199, 2016.

87. Grigorescu, S. M., B. Trasnea, T. T. Cocias and G. Macesanu, “A Survey of Deep Learning Techniques for Autonomous Driving”, *CoRR*, Vol. abs/1910.07738, 2019.

88. Wang, H., N. Wang and D. Yeung, “Collaborative Deep Learning for Recommender Systems”, *CoRR*, Vol. abs/1409.2944, 2014.

89. Zheng, L., V. Noroozi and P. S. Yu, “Joint Deep Modeling of Users and Items Using Reviews for Recommendation”, *CoRR*, Vol. abs/1701.04783, 2017.

90. Kendall, A. and Y. Gal, “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?”, *CoRR*, Vol. abs/1703.04977, 2017.

91. Kwon, Y., J.-H. Won, B. J. Kim and M. C. Paik, “Uncertainty Quantification Using Bayesian Neural Networks in Classification: Application to Biomedical Image Segmentation”, *Computational Statistics & Data Analysis*, Vol. 142, p. 106816, 2020.

92. Gal, Y. and Z. Ghahramani, “Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference”, *CoRR*, Vol. abs/1506.02158, 2015.

93. Zhou, Z.-H., *Ensemble Methods: Foundations and Algorithms*, Chapman & Hall/CRC, New York, 2012.

94. Matthey, L., I. Higgins, D. Hassabis and A. Lerchner, *dSprites: Disentanglement Testing Sprites Dataset*, 2017, `https://github.com/deepmind/dsprites-dataset/`, accessed in March 2021.

95. Sutskever, I., G. E. Hinton and G. W. Taylor, “The Recurrent Temporal Restricted Boltzmann Machine”, *Advances in Neural Information Processing Systems*, Vol. 21,

pp. 1601–1608, Curran Associates, Inc., 2008.

96. Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zhang, "Tensorflow: A System for Large-Scale Machine Learning", *CoRR*, Vol. abs/1605.08695, 2016.

APPENDIX A: NETWORK ARCHITECTURES

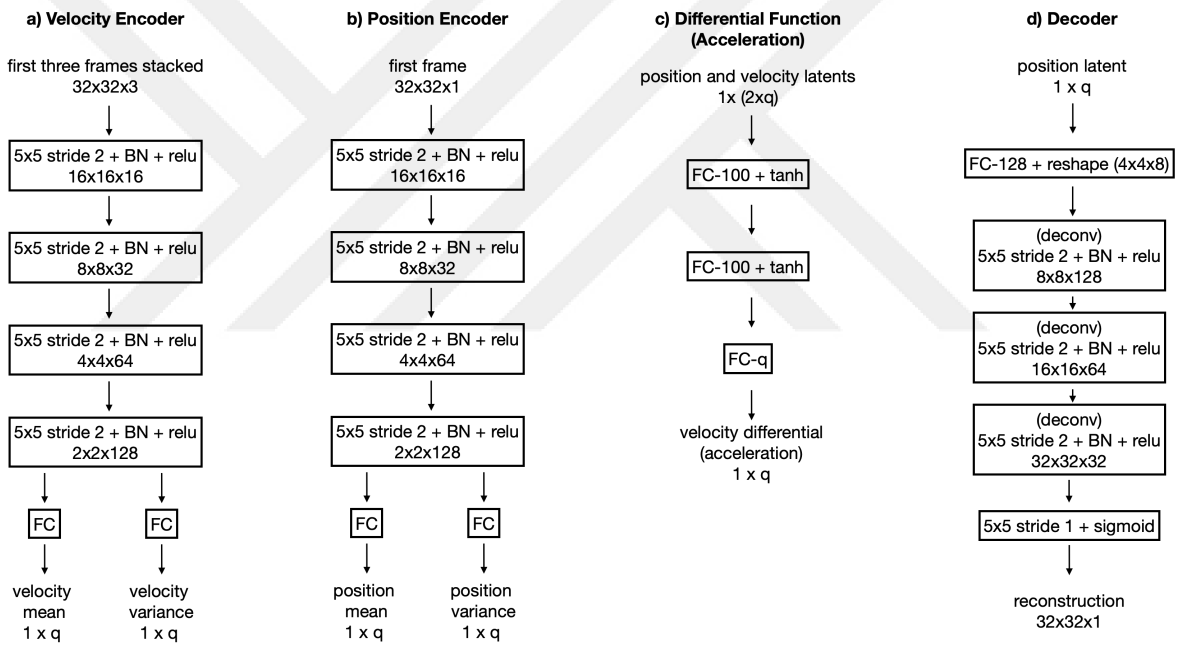

a) Velocity Encoder
first three frames stacked 32x32x3
5x5 stride 2 + BN + relu 16x16x16
5x5 stride 2 + BN + relu 8x8x32
5x5 stride 2 + BN + relu 4x4x64
5x5 stride 2 + BN + relu 2x2x128
FC
FC
velocity mean 1 x q
velocity variance 1 x q
b) Position Encoder
first frame 32x32x1
5x5 stride 2 + BN + relu 16x16x16
5x5 stride 2 + BN + relu 8x8x32
5x5 stride 2 + BN + relu 4x4x64
5x5 stride 2 + BN + relu 2x2x128
FC
FC
position mean 1 x q
position variance 1 x q
c) Differential Function (Acceleration)
position and velocity latents 1x (2xq)
FC-100 + tanh
FC-100 + tanh
FC-q
velocity differential (acceleration) 1 x q
d) Decoder
position latent 1 x q
FC-128 + reshape (4x4x8)
(deconv) 5x5 stride 2 + BN + relu 8x8x128
(deconv) 5x5 stride 2 + BN + relu 16x16x64
(deconv) 5x5 stride 2 + BN + relu 32x32x32
5x5 stride 1 + sigmoid
reconstruction 32x32x1

Figure A.1. Model scheme for ODE2VAE and $\beta-$ODE2VAE models. The figure is adapted from [5].

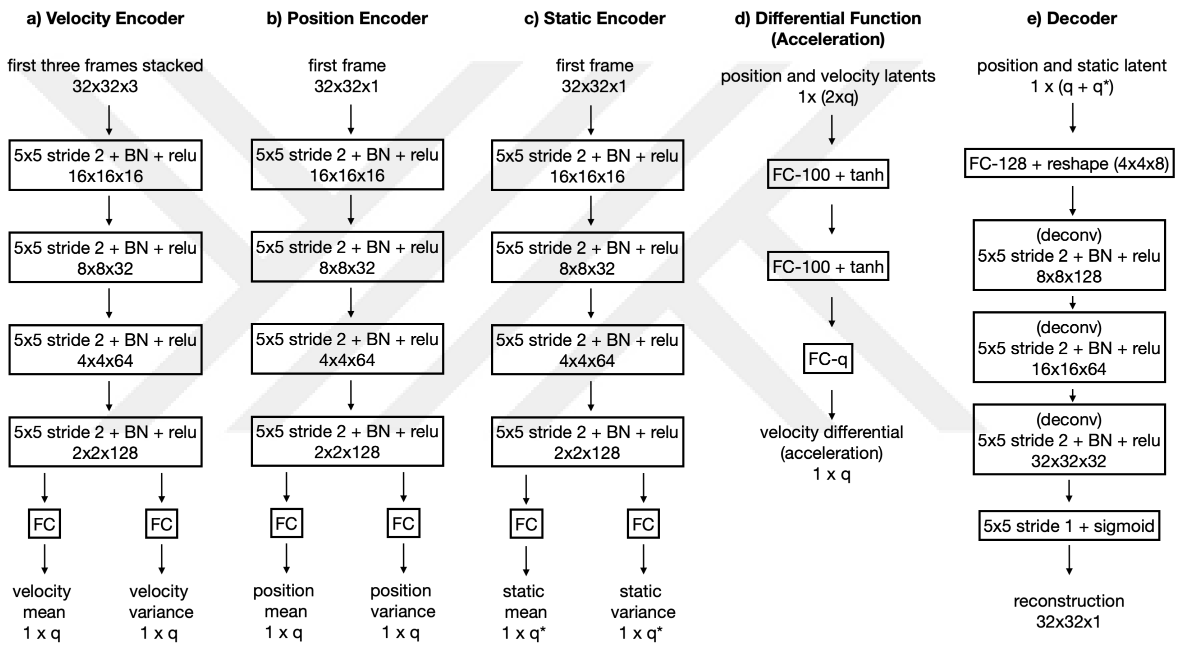

a) Velocity Encoder
first three frames stacked 32x32x3
5x5 stride 2 + BN + relu 16x16x16
5x5 stride 2 + BN + relu 8x8x32
5x5 stride 2 + BN + relu 4x4x64
5x5 stride 2 + BN + relu 2x2x128
FC
FC
velocity mean 1 x q
velocity variance 1 x q
b) Position Encoder
first frame 32x32x1
5x5 stride 2 + BN + relu 16x16x16
5x5 stride 2 + BN + relu 8x8x32
5x5 stride 2 + BN + relu 4x4x64
5x5 stride 2 + BN + relu 2x2x128
FC
FC
position mean 1 x q
position variance 1 x q
c) Static Encoder
first frame 32x32x1
5x5 stride 2 + BN + relu 16x16x16
5x5 stride 2 + BN + relu 8x8x32
5x5 stride 2 + BN + relu 4x4x64
5x5 stride 2 + BN + relu 2x2x128
FC
FC
static mean 1 x q*
static variance 1 x q*
d) Differential Function (Acceleration)
position and velocity latents 1x (2xq)
FC-100 + tanh
FC-100 + tanh
FC-q
velocity differential (acceleration) 1 x q
e) Decoder
position and static latent 1 x (q + q*)
FC-128 + reshape (4x4x8)
(deconv) 5x5 stride 2 + BN + relu 8x8x128
(deconv) 5x5 stride 2 + BN + relu 16x16x64
(deconv) 5x5 stride 2 + BN + relu 32x32x32
5x5 stride 1 + sigmoid
reconstruction 32x32x1

Figure A.2. Model scheme for ODE2VAE-c models. The encoders output the means and variances for the initial latent position and velocity states, and global static state. The figure is adapted from [5].

APPENDIX B: EXTRA RESULTS

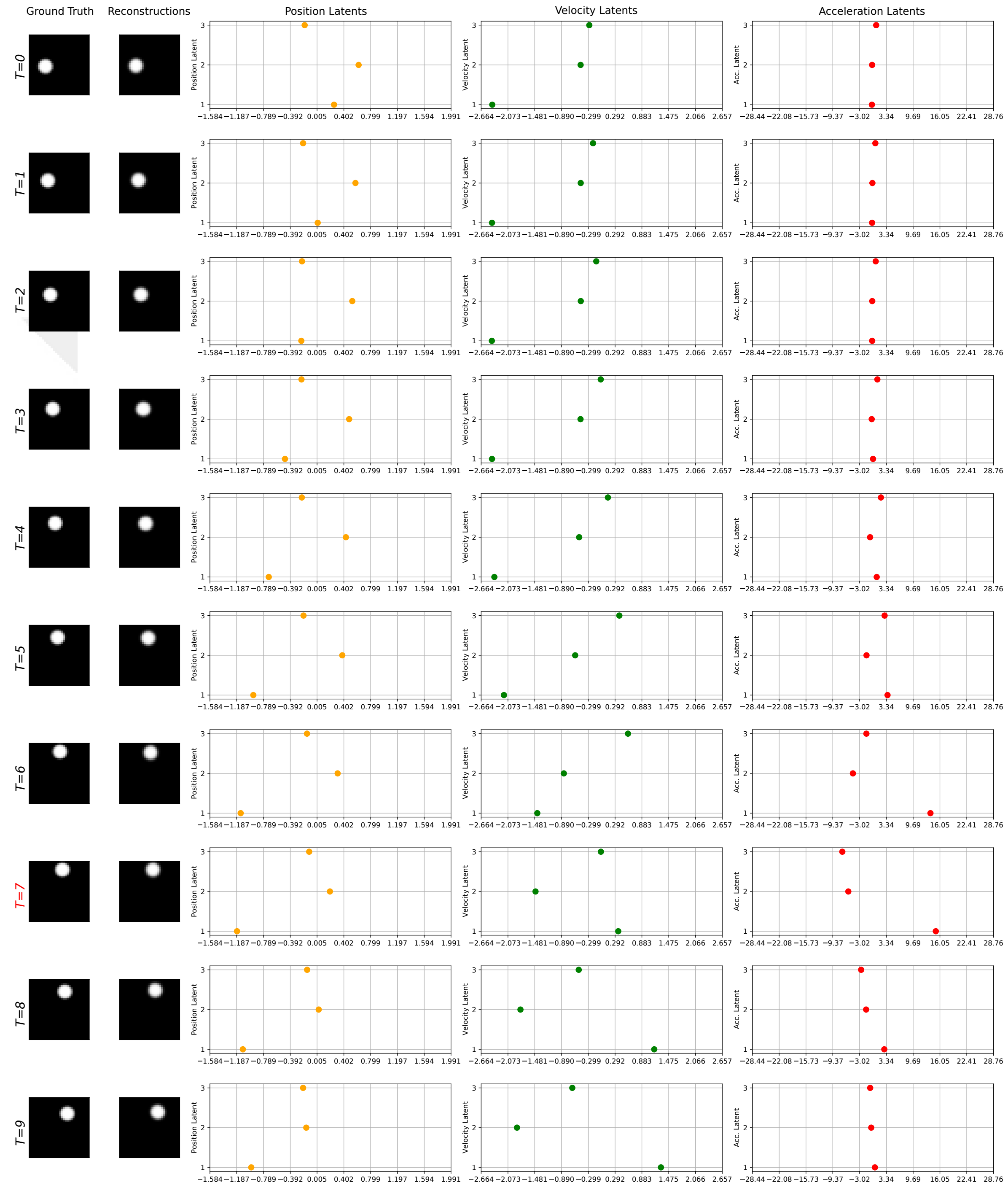


Figure B.1. Example test case for bouncing balls with $n = 1$ and corresponding latent vectors. The figure explicitly shows the evolution of the latent vectors.