# Deep Latent Variable Models for Sequential Data

Marco Fraccaro


Kongens Lyngby 2018

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

# Summary (English)

Over the last few decades an ever-increasing amount of data is being collected in a wide range of applications. This has boosted the development of mathematical models that are able to analyze it and discover its underlying structure, and use the extracted information to solve a multitude of different tasks, such as for predictive modelling or pattern recognition. The available data is however often complex and high-dimensional, making traditional data analysis methods ineffective in many applications. In the recent years there has then been a big focus on the development of more powerful models, that need to be general enough to be able to handle many diverse applications and kinds of data.

Some of the most interesting advancements in this research direction have been recently obtained combining ideas from probabilistic modelling and deep learning. Variational auto-encoders (VAEs), that belong to the broader family of deep latent variable models, are powerful and scalable models that can be used for unsupervised learning of complex high-dimensional data distributions. They achieve this by parameterizing expressive probability distributions over the latent variables of the model using deep neural networks. VAEs can be used in applications with static data, for example as a generative model of images, but they are not suitable to model temporal data such as the sequences of images that form a video. However, a major part of the data that is being collected has a sequential nature, and finding powerful architectures that are able to model it is therefore fundamental.

In the first part of the thesis we will introduce a broad class of deep latent variable models for sequential data, that can be used for unsupervised learning of complex and high-dimensional sequential data distributions. We obtain these models by extending VAEs to the temporal setting, and further combining ideas from deep learning (e.g. deep and recurrent neural networks) and probabilistic modelling (e.g. state-space models) to define generative models for the data that use deep neural networks to parameterize very flexible probability distributions. This results in a family of powerful architectures that can model a wide range of complex temporal data, and can be trained in a scalable way using large unlabelled datasets.

In the second part of the thesis we will then present in detail three architectures belonging to this family of models. First, we will introduce *stochastic recurrent neural networks* (Fraccaro et al., 2016c), that combine the expressiveness of recurrent neural networks and the ability of state-space models to model the uncertainty in the learned latent representation. We will then present

*Kalman variational auto-encoders* (Fraccaro et al., 2017), that can learn from data disentangled and more interpretable visual and dynamic representations. Finally, we will show that to deal with temporal applications that require a high memory capacity we can combine deep latent variable models with external memory architectures, as in the *generative temporal model with spatial memory* of Fraccaro et al., (2018).

# Resumé (Summary in Danish)

I løbet af de sidste par årtier er der samlet en stadig større mængde data i en bred vifte af anvendelser. Dette har styrket udviklingen af matematiske modeller, der kan analysere og opdage den underliggende struktur i data, og bruge den uddragne information til at løse en lang række forskellige opgaver, f.eks. til prediktiv modellering eller mønstergenkendelse. De tilgængelige data er imidlertid ofte komplekse og høj dimensionelle, hvilket ofte gør traditionelle dataanalysemetoder ineffektive. I de senere år har der været et stort fokus på udviklingen af mere kraftfulde modeller, der skal være generelle nok til at kunne håndtere mange forskellige typer anvendelser og data.

Nogle af de mest interessante fremskridt i denne forskningsretning er for nylig blevet opnået ved at kombinere ideer fra probabilistisk modellering og dyb læring. Variational auto-encoders (VAE'er), der tilhører den bredere familie af dybe latente variabel modeller, er fleksible og skalerbare modeller, som kan bruges til uovervåget (eng unsupervised) læring af komplekse høj dimensionelle datafordelinger. De opnår dette ved at parameterisere ekspressive sandsynlighedsfordelinger over de latente variabler ved hjælp af dybe neurale netværk. VAE'er kan bruges i applikationer med statiske data, for eksempel som en generativ model af billeder, men de er ikke egnede til at modellere tidsmæssige data, såsom sekvenserne af billeder (video). En stor del af de data, der indsamles, har imidlertid en sekventiel karakter, og det er derfor en fundamental udfordring at finde ekspressive arkitekturer, der kan modellere det.

I den første del af afhandlingen introducerer vi en bred klasse af dyb latent variabel modeller til sekventielle data, der kan bruges til uovervåget læring af komplekse og høj dimensionelle sekventielle datafordelinger. Vi kommer frem til disse modeller ved at udvide VAE'erne til det tidslige domæne og yderligere kombinere ideer fra dyb læring f.eks. dybe og rekursive neurale netværk (eng recurrent neural networks) og probabilistisk modellering f.eks. state-space-modeller. Dette resulterer i en familie af arkitekturer, som kan bruges til at modellere en bred vifte af komplekse tidsmæssige data og kan trænes på en skalerbar måde ved hjælp af store datasæt.

I anden del af afhandlingen vil vi så detaljeret præsentere tre arkitekturer af ovennævnte type. For det første vil vi introducere *stochastic recurrent neural networks* (Fraccaro et al., 2016c), der kombinerer ekspressiviteten af rekursive neurale netværk og state-space-modellers evne til at modellere usikkerheden i den lærte latente repræsentation. Vi vil derefter præsentere *Kalman variational auto-encoders* (Fraccaro et al., 2017), der kan lære afkoblede og mere fortolkelige visuelle og dynamiske data repræsentationer. Endelig vil vi vise at for at håndtere anvendelser på

temporal data, der kræver en høj hukommelseskapacitet, kan vi kombinere dyb latente variabel modeller med eksterne hukommelsesarkitekturer, som i *generative temporal model with spatial memory* af Fraccaro et al., (2018).

# List of Publications

## Contributions included in this thesis

**(Fraccaro et al., 2016c)** Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet and Ole Winther. *Sequential Neural Models with Stochastic Layers.* Advances in Neural Information Processing Systems 29, NIPS 2016.

**(Fraccaro et al., 2017)** Marco Fraccaro, Simon Kamronn, Ulrich Paquet and Ole Winther. *A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning.* Advances in Neural Information Processing Systems 30, NIPS 2017.

**(Fraccaro et al., 2018)** Marco Fraccaro, Danilo Rezende, YoriZwols, Alexander Pritzel, Ali Eslami and FabioViola. *Generative Temporal Models with Spatial Memory for Partially Observed Environments.* Proceedings of the 35th International Conference on Machine Learning, ICML 2018.

## Contributions not included in this thesis

**(Opper et al., 2015)** Manfred Opper, Marco Fraccaro, Ulrich Paquet, Alex Susemihl and Ole Winther. *Perturbation Theory for Variational Inference.* NIPS Workshop on Advances in Approximate Bayesian Inference, 2015.

**(Fraccaro et al., 2016b)** Marco Fraccaro, Ulrich Paquet and Ole Winther. *Indexable Probabilistic Matrix Factorization for Maximum Inner Product Search.* The Thirtieth AAAI Conference on Artificial Intelligence, 2016.

**(Fraccaro et al., 2016a)** Marco Fraccaro, Ulrich Paquet and Ole Winther. *An Adaptive Resample-Move Algorithm for Estimating Normalizing Constants.* ArXiv preprint, 2016.

**(Paquet and Fraccaro, 2016)** Ulrich Paquet and Marco Fraccaro. *An efficient implementation of Riemannian Manifold Hamiltonian Monte Carlo for Gaussian Process Models.* Technical report, 2016.

**(Maaløe et al., 2017)** Lars Maaløe, Marco Fraccaro and Ole Winther. *CaGeM: A Cluster Aware Deep Generative Model.* NIPS Workshop on Advances in Approximate Bayesian Inference, 2017.

# Preface

This thesis was prepared at the Cognitive Systems section of DTU Compute, Department of Applied Mathematics and Computer Science, Technical University of Denmark. It constitutes a partial fulfillment of the requirements for acquiring a Ph.D. at the Technical University of Denmark.

This PhD project was financed by Microsoft Research through its PhD Scholarship Programme and by DTU Compute, and was supervised by Ole Winther (DTU Compute) and Ulrich Paquet (Microsoft Research, now at Google DeepMind). The PhD project was carried out at DTU during the period October 2014 - April 2018, except for two three-months leaves of absence taken for internships at Microsoft Research Cambridge (summer 2015, supervised by Tom Minka) and Google DeepMind (summer 2017, supervised by Danilo Rezende).

During the course of my PhD I have worked on many different topics in recommender systems, Bayesian inference, sequential Monte Carlo methods and deep learning, that resulted in six peer-reviewed publications and two preprints. However, this thesis only focuses on the work on deep latent variable models for sequential data that was done during the second half of the PhD.

The thesis is divided in two main parts. The first part consists of an introduction to a broad family of models for sequential data, while the second part contains three research papers (two published, one currently under review) that describe in detail three architectures belonging to this family of models.

Lyngby, 13-April-2018

Marco Fraccaro

# Acknowledgements

The results presented in this thesis would not have been possible without the support, collaborations, discussions and ideas of many different people.

First of all, I am deeply grateful to my supervisors Ole Winther and Ulrich Paquet for their fundamental role in everything I have accomplished in the last three and a half years. Your inspiration and enthusiasm have guided me through a challenging but extremely fun journey I can be proud of! I truly hope that even after my PhD I can keep learning from you and collaborate in new exciting projects. A special thanks to Ulrich for his help in writing papers until late at night before every deadline, as well as his PR skills that were fundamental to get the funding for this PhD project and my internships.

During my Master and PhD I spent most of my time at the Cognitive Systems (CogSys) section of DTU Compute, an amazing place whose courses and professors transformed me from a newbie in Machine Learning to a researcher that masters it well enough to publish and obtain a PhD. I wish to thank all my CogSys colleagues for the many great discussions, and in particular my collaborators, the deep learners and everyone from the Bayesian reading/cake group! During my PhD I was lucky enough to intern with two amazing researchers: Tom Minka (Microsoft Research Cambridge) and Danilo Rezende (Google DeepMind). Thank you both for you support and guidance during the internship, it has been lots of fun! Also, thanks to all the other great Microsoft and Google researchers and interns I have met, that made these experiences even better!

I am grateful for the financial support from Microsoft Research and DTU Compute for my PhD studies, Otto Mønsteds Fond for allowing me to travel to many different conferences all over the world, and Nvidia for donating many of the GPUs I used to run experiments.

Last but not least, I am grateful to my friends and family in Denmark, Italy, Colombia and many other parts of the world, for the fantastic moments spent together. A huge thanks to my parents Mario and Giovanna, my sister Claudia and my girlfriend Martina who supported me with love over all these years, and were kind enough to bear with me while I was busy and working hard before the many exams and deadlines I had during the course of my studies.

# Contents

# Part I

# Introduction

Chapter 1

# Introduction

## 1.1 Motivation

The majority of the successful practical applications of machine learning use *supervised learning* methods, that learn a mapping from an input to an output variable using a dataset containing *labelled data*. We may want for example to learn a classification model that receives an image of an object as input, and outputs a label describing the type of object in the image. For training this model we require a labelled dataset that contains a set of image-label pairs from which the model can learn to distinguish the different objects. However, labelled data is scarce and expensive to obtain, in contrast to the vast amount of *unlabelled data* collected daily in the Web and in the smart devices connected to the Internet of Things (IoT).

A major focus in recent machine learning research is therefore the development of *unsupervised learning* methods, that use the available unlabelled data. In unsupervised learning, the interest is in learning models that can describe the underlying structure of the data, e.g. interesting patterns, clusters, statistical correlations or causal structures. We can use it for example to uncover the hidden structures in a collection of images, learn how to predict some of the pixels of an image given the rest of them (e.g. to deal with occlusions) or generate new images from the same data distribution.

An unsupervised model that explains the data well, can also be used for *semi-supervised* learning, in which we want to solve a supervised task given a small labelled dataset and a big unlabelled one. This is particularly relevant for applications in which obtaining a labelled dataset is difficult or expensive but there is a large availability of unlabelled data. Semi-supervised learning methods use the unlabelled data to learn a latent representation of the inputs that makes the supervised task simpler, and typically achieve better performances than the ones obtained with a model solely trained on the small labelled dataset.

Unsupervised learning is therefore relevant and applicable in many contexts and, since it does

not require labelled data, training can be performed using very large datasets. It is generally much harder than supervised learning, as instead of predicting a single label given the input, unsupervised methods have learn to describe the structure of the input itself. In the last few years there have been major advancements in unsupervised learning of complex high-dimensional data distributions such as images, most of which build on the seminal works on variational auto-encoders (VAEs) (Kingma and Welling, 2014; Rezende et al., 2014) and generative adversarial networks (GANs) (Goodfellow et al., 2014). Broadly speaking, these models combine ideas from deep learning and probabilistic modelling, by defining generative models for the data that use deep neural networks to parameterize very expressive probability distributions. This results in complex but flexible models containing hundreds of thousands of parameters, that are trained in a scalable way using large datasets of unlabelled data and exploiting the computational capabilities of graphics processing units (GPUs).

Most of the focus in recent unsupervised learning research has been on *static* data, i.e. fixed data with no sequential nature such as images. A big part of the available unlabelled data is sequential: we may be interested for example in modelling videos (a sequence of images), speech, music, text, visits in electronic health records, the evolution of financial markets, user-click behaviors or sensor data, all of which have an inherent temporal component. State-of-the-art models for unsupervised learning of high-dimensional distributions such as VAEs are however difficult to use in the sequential setting, as they cannot properly model temporal correlations in the data.

The aim of this thesis is to introduce a general class of sequential models for unsupervised learning that can be used to model a wide range of complex temporal data and can be trained in a scalable way using large unlabelled datasets. We combine deep learning architectures with probabilistic models for sequential data that use deep neural networks to parameterize their distributions, therefore obtaining flexible models inspired by VAEs. This will be discussed more in detail in Section 1.2. These models can be used for a broad range of tasks such as for generative modelling, representation learning, predictive modelling, compression, probabilistic reasoning or planning in model-based reinforcement learning.

The primary target audience of this thesis is practitioners who are interested in flexible models for unsupervised learning of high-dimensional sequential data distributions. We assume some experience in basic probability theory and deep learning.

## 1.2 Outline and contributions

Despite its importance, the progress on unsupervised learning in the sequential setting has been slower than the one for the static case. One of the main challenges when learning such methods for sequential data is that the ideas they build on were developed in many different scientific areas such as control theory, aerospace engineering, econometrics, statistics and machine learning (both in probabilistic modelling and deep learning), each of which uses different terminologies and notations. In the first half of this thesis (Chapters 2 to 4) we then present a *unified treatment* of these topics from a machine learning perspective, hoping to help the reader understand the "big picture" of how all these seemingly disparate ideas and models are connected. We also introduce simple examples and intuitions that can be helpful to grasp the rationale behind more

complex methods[1]. The second half of the thesis (Chapters 5 to 7) contains three publications that present some of the novel models for unsupervised learning for high dimensional temporal data distributions, briefly discussed in Chapter 4, in greater detail.

**Chapter 2** introduces *latent variable models* (LVMs), the building blocks of all the models presented in the rest of the thesis, and shows how they can be used for unsupervised learning in the static setting. LVMs are probabilistic models that use a *latent variable* to model the generative process from which the data was created and capture its hidden structure. We show that in many cases the integrals needed during posterior inference and parameter learning are not analytically tractable, but can be approximated using Variational inference. We then introduce a flexible and scalable architecture, called variational auto-encoder, that can model a wide range of data distributions by using the function approximation capabilities of deep neural networks to parameterize the probability distributions that define it. Thanks to the deep neural networks, this model can learn to automatically extract useful features in a hierarchical way.

**Chapter 3** presents *state-space models* (SSM) as sequential extensions to LVMs, that provide a general framework for sequential data modelling. We show that depending on our modelling assumptions we can define different classes of SSMs. The linear Gaussian state-space model is a basic architecture that makes strong assumptions on the generative process of the data, but for which posterior inference is analytically tractable. It is also possible to define more expressive non-linear and non-Gaussian architectures, for which however we will need to perform approximate posterior inference building on the techniques discussed in Chapter 2 for the static setting.

**Chapter 4** shows that we can combine the ideas presented in Chapters 2 and Chapter 3 to define a general family of methods for large-scale unsupervised learning of complex high-dimensional data distributions in the sequential setting. We build a wide variety of models merging ideas from deep learning (e.g. recurrent neural networks and external memories) and probabilistic modelling (e.g. state-space models and variational auto-encoders). These models can be trained in a scalable way using *amortized inference* ideas from VAEs, with black-box methods that allow us to focus more on the modelling side than on the inference one. However, we also show that inference can be further improved by tailoring the posterior approximation to the specific model. This chapter contains the most novel component of the unified treatment presented in the first half of the thesis, as we illustrate the strong connections between many of the models presented in the recent literature as well as between the approximate inference techniques used for training.

**Chapter 5** contains the paper (Fraccaro et al., 2016c), that introduces *stochastic recurrent neural networks* (SRNN). For some sequential datasets recurrent neural networks (RNNs) are not enough, as they cannot model the stochasticity in the latent representation. We then define a model that combines the ability of RNNs to capture long-term dependencies in the data and the ability of state-space models to model the uncertainty in the stochastic latent states. We use flexible state-space models parameterized by deep networks, that can be trained extending ideas from variational autoencoders to the temporal setting as shown in Chapter 4.

**Chapter 6** contains the paper (Fraccaro et al., 2017), that introduces *Kalman variational auto-encodes* (KVAE). Most of the models presented in this thesis are expressive but black-box

[1] In other words, I have collected in this thesis all the knowledge I wish I had before starting to work on these topics.

architectures, therefore their results are difficult to interpret. In this paper we show that by carefully designing the model including some domain knowledge in a structured prior distribution we can learn disentangled visual and dynamic representations, that make the model more interpretable and computationally efficient when making predictions for future time steps. The backbone of the KVAE is given by a linear Gaussian state-space model, and we can therefore exploit its exact inference procedures and missing data imputations capabilities.

**Chapter 7** contains the paper (Fraccaro et al., 2018), that introduces *generative temporal models with spatial memory* (GTM-SM). In some applications that require a big memory capacity recurrent neural networks are not powerful enough to memorize all the needed information. This is the case for example when modelling agents walking in an environment that need to remember what they have seen in the past. In this paper we show that this task can be solved by combining a structured prior similar to the one presented in Chapter 6 with a non-parametric differentiable external memory architecture.

**Chapter 8** finally summarizes the main contributions of this thesis, discusses open questions and some directions for future work.

Chapter 2

# Latent Variable Models

## 2.1 Latent variable models

One of the central problems in machine learning is that of unsupervised learning of complicated probability distributions $p(\mathbf{x})$ given a finite sample of possibly high-dimensional data points $\mathbf{x}$ drawn from that distribution. For example, consider the task of learning a probability distribution over images of houses. To be able to do this, we need to define a distribution that can model the complex correlations between the hundreds of thousands of pixels that form each image, that are due to the recurring textures and patterns in the image. For example, neighboring pixels will likely have similar colors and there will be multiple windows of the same type. To build such distributions $p(\mathbf{x})$ that are both flexible and scalable enough, several methods cast the image modelling problem to a sequential one, defining an ordering for the pixels in the image and learning to to predict the next pixel given all the previous ones (Larochelle and Murray, 2011; Van Den Oord et al., 2016). However, due to the high number of pixels in a sequence, capturing the correlations between distant pixels in the image is challenging.

Rather than modelling the distribution $p(\mathbf{x})$ directly, we can introduce an unobserved *latent variable* $\mathbf{z}$ and define a conditional distribution $p(\mathbf{x}|\mathbf{z})$ for the data, also known as *likelihood*. In the following, we assume that $\mathbf{z}$ is a continuous random variable, but many of the ideas described below also apply to the discrete case. Each of the elements of the observed variable $\mathbf{x}$ will depend on the latent variable $\mathbf{z}$, that can therefore be used to express the correlations in the observed variable $\mathbf{x}$. When modelling images of houses, $\mathbf{z}$ could for example contain a latent representation of the type of house in the image, its architectural style, the color of the walls and so on. We introduce a *prior distribution* $p(\mathbf{z})$ over the latent variables, and compute the *joint distribution* over observed and latent variables as

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}) \ . \tag{2.1}$$

The introduction of the latent variable in the model allows us to express the complex marginal distribution $p(\mathbf{x})$ in terms of a more tractable joint distribution, whose components $p(\mathbf{x}|\mathbf{z})$ and

$p(\mathbf{z})$ are typically much simpler to define, for example by using exponential family distributions. Given the joint distribution we obtain the desired data distribution $p(\mathbf{x})$ by marginalizing over the latent variables:

$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z} = \int p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) \mathrm{d}\mathbf{z} \ . \tag{2.2}$$

Using Bayes' rule we can then compute the *posterior distribution* $p(\mathbf{z}|\mathbf{x})$ as

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{p(\mathbf{x})} \ , \tag{2.3}$$

which allows us to *infer* the latent variable given the observation.

Models with latent variables can be interpreted as expressing the generative process from which the data was created: to generate a new data point we first get a sample $\mathbf{z}^{(s)}$ from $p(\mathbf{z})$, and we then use it to sample a new observation $\mathbf{x}^{(s)}$ from the conditional distribution $p(\mathbf{x}|\mathbf{z}^{(s)})$. Samples obtained in this way can also be used to asses whether the model provides a good approximation to the data distribution. In our example, $\mathbf{z}$ can be interpreted as containing a latent representation of the architectural choices that where taken when designing the house, that condition the house that is actually built, i.e. the observation $\mathbf{x}$.

The specific relationship between the latent variable $\mathbf{z}$ and the observation $\mathbf{x}$ depends on the form of the distributions in (2.1). Often we assume that the likelihood $p(\mathbf{x}|\mathbf{z})$ and the prior $p(\mathbf{z})$ belong to parametric families of distributions. To make this explicit in the notation, we can write the joint distribution as

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) \ , \tag{2.4}$$

where $\theta$ denotes the unknown parameters of the model, that can be learned from data as discussed in detail in Section 2.3.

Using latent variables that have a much lower dimensionality than the observed vectors we can obtain a compressed representation of the data. In this case in fact, the latent variables act as as a bottleneck through which all the information needed to generate the observations has to pass. This is justified by the fact that in many data sets the data lies in a manifold whose dimensionality is much smaller than the one of the original data space. Latent variable models can be used as black-box density models, but we can also include some prior knowledge on the generative mechanism that created the data in the distributions that define the joint $p(\mathbf{x}, \mathbf{z})$ e.g. using probabilistic graphical models. We will return to this point in Section 3.1 when discussing state-space models.

Mathematical models containing latent variables are defined as *latent variable models* (LVMs). Among this class of methods we find *linear Gaussian models* such as factor analysis, principal component analysis and mixture of Gaussians, as presented in the seminal paper by Roweis and Ghahramani, (1999). These models have the advantage that posterior inference is tractable, but are not expressive enough to model the kind of high-dimensional data we are interested in. This chapter will then focus on *non-linear* LVMs, that are more suitable to model complex high-dimensional data distributions but that require approximate inference as for them the integral in (2.2) has no analytic solution. In Section 2.4 we will introduce *variational auto-encoders* (VAEs), that merge ideas from deep learning and latent variable models by using deep neural networks to define very flexible distributions for (2.1). We broadly define such non-linear LVMs in which the non-linearities are given by deep neural networks as *Deep Latent Variable Models* (DLVMs).

Before introducing VAEs, the next sections present scalable techniques based on variational methods to perform approximate inference and learning in non-linear LVMs.

## 2.2 Posterior inference

The posterior distribution in (2.3) represents our updated beliefs about the latent variables after having seen the data, and it is a key component for probabilistic reasoning in LVMs. In many of the models presented in this thesis, however, the posterior is intractable due to the lack of an analytic solution to the integral in (2.2) that appears in the denominator of (2.3). There are broadly two classes of methods that were developed to approximate the posterior distribution. They trade accuracy of the approximation with computational time:

1. *Sampling techniques* such as Markov Chain Monte Carlo (MCMC) methods provide a sample-based approximation to the posterior distribution. In most cases the posterior is needed primarily to evaluate expectations, that can be approximated using the posterior samples with Monte Carlo integration. Sampling methods have the appealing property that given infinite computational resources they generate exact results, and the approximation only comes from the fact that we have a limited amount of resources in practice. However, these methods are in general computationally expensive and do not scale well to large data sets. Furthermore, it is often difficult to diagnose their convergence.

2. *Deterministic approximation techniques* are based on analytic approximations to the posterior distribution, where we assume for example that the posterior comes from a particular parametric family of distributions or that it factorizes in a certain way. These are very scalable methods, but even given infinite computational resources they cannot generate exact results. Among this class of methods we find for example the Laplace approximation, variational inference and expectation propagation.

This thesis focuses on large data sets of high-dimensional data, for which variational inference provides a good trade-off between quality of the approximation and scalability of the inference procedure.

### 2.2.1 Variational inference

In *variational inference* we use the calculus of variations to find the posterior approximation $q(\mathbf{z})$ that minimizes a measure of dissimilarity between $q(\mathbf{z})$ and the true posterior $p(\mathbf{z}|\mathbf{x})$. While there are many different ways to measure of how different two distributions are, variational inference uses the *Kullback–Leibler (KL) divergence* between the variational distribution and the posterior distribution[1], defined as

$$\mathrm{KL}\left[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right] = -\mathbb{E}_{q(\mathbf{z})}\left[\log \frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})}\right] , \tag{2.5}$$

[1] Variational inference can also be used with other divergences, see for example (Li and Turner, 2016).

where $\mathbb{E}_{q(\mathbf{z})}$ denotes an expectation over $q(\mathbf{z})$. Importantly, the KL divergence is a non-negative quantity, i.e. $\mathrm{KL}\left[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right] \geq 0$, with equality if and only if $q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x})$. The more similar $q(\mathbf{z})$ is to $p(\mathbf{z}|\mathbf{x})$, the smaller the KL divergence will be. Notice that this quantity is however not a *distance* in the mathematical sense, as it is not symmetric if we swap the two distributions.

Our goal is to find a good variational approximation $q(\mathbf{z})$ that minimizes the KL divergence in (2.5). However, this quantity is still not tractable as the intractable posterior $p(\mathbf{z}|\mathbf{x})$ appears at the numerator inside the logarithm. Using (2.3) we can rewrite (2.5) as

$$\begin{aligned}\mathrm{KL}\left[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right] &= -\mathbb{E}_{q(\mathbf{z})}\left[\log\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})} - \log p(\mathbf{x})\right] \\ &= \underbrace{-\mathbb{E}_{q(\mathbf{z})}\left[\log\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]}_{\mathcal{F}(q)} + \log p(\mathbf{x}) \ , \end{aligned} \tag{2.6}$$

where the marginal likelihood $\log p(\mathbf{x})$ could be taken out of the expectation as it is independent from $\mathbf{z}$. The quantity $\mathcal{F}(q)$ is known as *Evidence Lower BOund* (ELBO), as due to the non-negativity of the KL divergence it represents a lower bound to the *evidence* $\log p(\mathbf{x})$, i.e. $\log p(\mathbf{x}) \geq \mathcal{F}(q)$ for all $q(\mathbf{z})$. Notice that in (2.6) the numerator inside the logarithm is now tractable, since it consists of the joint distribution in (2.1), and $\log p(\mathbf{x})$ is constant for all $q(\mathbf{z})$. This means that to minimize $\mathrm{KL}\left[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right]$, and finding therefore the optimal variational approximation, we can just maximize the ELBO with respect to the distribution $q(\mathbf{z})$: the closer the ELBO is to the marginal likelihood, the closer (in KL sense) the variational approximation will be to the posterior distribution. Using variational methods, we can therefore reduce a complex inference problem to a simpler optimization problem.

In practice, the *variational distribution* $q(\mathbf{z})$ is often restricted to a particular parametric family for which the ELBO is tractable or simple to approximate (e.g. Gaussian distribution), and the maximization with respect to $q(\mathbf{z})$ is therefore a maximization with respect to the parameters of the family. The family of distribution $q(\mathbf{z})$ needs to be flexible enough to provide a good approximation to the posterior distribution, but simple enough to make the ELBO easy to compute. We will return to this point when introducing the posterior approximation for the VAE in Section 2.5.2.

## 2.3 Parameter learning

We assume that the likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ and the prior $p_\theta(\mathbf{z})$ belong to families of distributions that depend on some unknown parameters $\theta$. Given a training set with $N$ data points $\{\mathbf{x}^i\}_{i=1}^N$, the optimal parameters $\theta^\star$ of the model can be learned using Maximum Likelihood Estimation (MLE), i.e. maximizing

$$\begin{aligned}\mathcal{L}(\theta) &= \sum_{i=1}^{N} \log p_\theta(\mathbf{x}^i) \\ &= \sum_{i=1}^{N} \underbrace{\log \int p_\theta(\mathbf{x}^i, \mathbf{z}^i)\mathrm{d}\mathbf{z}^i}_{\mathcal{L}_i(\theta)}\end{aligned}$$

with respect to the parameters $\theta$ of the model. Notice in particular that while we have a different latent variable $\mathbf{z}^i$ for each data point $\mathbf{x}^i$, the parameters $\theta$ of the likelihood and prior are shared across all data points. In this thesis, we assume that the parameters are fixed but unknown quantities. Alternatively, one could follow a Bayesian approach, considering them as random variables with a prior distribution $p(\theta)$ and working with the joint distribution $p(\mathbf{x}^i, \mathbf{z}^i, \theta) = p(\mathbf{x}|\mathbf{z}^i, \theta)p(\mathbf{z}^i|\theta)p(\theta)$. In the rest of this chapter, we will omit the superscript $i$ when only one data point is referred to, or when it is clear from the context.

As discussed in the previous sections, in many cases the marginal density of the observations $p_\theta(\mathbf{x})$ is intractable and needs to be approximated. As we have seen in Section 2.2.1, a possible approximation can be obtained using the ELBO. This result can be re-derived in an alternative way, provided below as it is widely used in the literature and gives interesting insights in variational methods. For any distribution $q(\mathbf{z})$ over the latent variables, we can compute a lower bound to $\log p_\theta(\mathbf{x})$ as follows:

$$\begin{aligned}
\mathcal{L}_i(\theta) = \log p_\theta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z} \\
&= \log \int \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} q(\mathbf{z}) \mathrm{d}\mathbf{z} \\
&= \log \mathbb{E}_{q(\mathbf{z})} \left[ \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \\
&\geq \mathbb{E}_{q(\mathbf{z})} \left[ \log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] = \mathcal{F}_i(\theta, q) \ , \qquad (2.7)
\end{aligned}$$

where we have used the concavity of the logarithm and Jensen's inequality to swap the logarithm and the expectation in the last line. $\mathcal{F}_i(\theta, q)$ is exactly the ELBO introduced in (2.6), but we extended the notation to stress the dependence on the parameters $\theta$ of the model and on data point $\mathbf{x}^i$. We have then shown in an alternative way that the ELBO is a lower bound to $\log p(\mathbf{x})$, i.e. $\mathcal{L}_i(\theta) \geq \mathcal{F}_i(\theta, q)$. To learn the parameters of the model, instead of maximizing the log-likelihood $\mathcal{L}(\theta)$, we can then maximize the total ELBO $\mathcal{F}(\theta, q) = \sum_{i=1}^N \mathcal{F}_i(\theta, q)$ with respect to $\theta$ and $q(\mathbf{z})$. As we have seen in Section 2.2.1, the variational distribution can be interpreted as an approximation to the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$, and the ELBO coincides with the log-likelihood if and only if $q(\mathbf{z})$ is the posterior distribution, i.e. $q(\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x})$.

### 2.3.1 The Expectation Maximization (EM) algorithm

The *Expectation Maximization* (EM) algorithm is a two-stage iterative optimization method for MLE of the parameters of a model with latent variables in it (Dempster et al., 1977). The EM algorithm can be formulated in its most general form starting from the ELBO in (2.7). It alternates between two steps, that maximize $\mathcal{F}_i(\theta, q)$ with respect to $q(\mathbf{z})$ and $\theta$ respectively, while holding the other fixed. We start from some parameters $\theta_0$, and until convergence we repeat

$$\textbf{E-step:} \qquad q_{k+1} = \operatorname*{argmax}_q \mathcal{F}_i(\theta_k, q) \qquad (2.8)$$

$$\textbf{M-step:} \qquad \theta_{k+1} = \operatorname*{argmax}_\theta \mathcal{F}_i(\theta, q_{k+1}) \qquad (2.9)$$

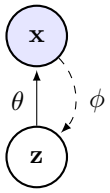



***Figure 2.1:*** *Graphical model of a variational auto-encoder. Here and in the other graphical models in this thesis empty nodes denote latent variables, while colored ones denote observations.*

For many models, each of these step will be simpler than updating both $q(\mathbf{z})$ and $\theta$ at the same time (in Section 2.4.3 however, we will see that for variational auto-encoders it is easy to perform joint maximization). As in the E-step we are holding the parameters of the model fixed, this step is basically solving a posterior inference problem as the one introduced in Section 2.2.1, therefore the optimal distribution is $q_{k+1}(\mathbf{z}) = p_{\theta_k}(\mathbf{z}|\mathbf{x})$. For models in which the posterior $p_{\theta_k}(\mathbf{z}|\mathbf{x})$ in the E-step is intractable, we can do a partial optimization of $q(\mathbf{z})$, i.e. approximate inference. In the M-step we then fix the distribution over the latent variables and we maximize the ELBO with respect to the parameters $\theta$ of the model, using for example gradient ascent methods.

Interestingly, for simpler classes of models for which inference is exact, we are guaranteed not to decrease the marginal likelihood after each combined EM step, i.e. $\mathcal{L}_i(\theta_{k+1}) \geq \mathcal{L}_i(\theta_k)$. After the E-step, that does not change the value of $\mathcal{L}_i(\theta_k)$ as $\theta_k$ is held fixed, we have $\mathcal{L}_i(\theta_k) = \mathcal{F}_i(\theta_k, q)$. The subsequent maximization of $\mathcal{F}_i(\theta, q_{k+1})$ in the M-step will therefore not decrease $\mathcal{L}_i(\theta_{k+1})$, that is lower bounded by $\mathcal{F}_i(\theta, q_{k+1})$.

## 2.4 Variational auto-encoders

*Variational auto-encoders* (VAEs) (Kingma and Welling, 2014; Rezende et al., 2014) use deep neural networks to parameterize the probability distributions that define a latent variable model, while providing an efficient approximate inference procedure that scales to large dataset with millions of data point. VAEs have recently had a huge impact in several communities. In the deep learning community, VAEs are mostly being used as generative models of high-dimensional data, e.g. to generate artificial images that resemble real ones (Gulrajani et al., 2016; Chen et al., 2017). The focus in the probabilistic modelling community, on the other hand, is that of extending to many different probabilistic models ideas first introduced with VAEs, i.e. using deep neural networks to define flexible probability distributions while retaining scalable inference (Johnson et al., 2016; Fraccaro et al., 2016c; Krishnan et al., 2017). This thesis fits in the latter category, as these ideas are used to define expressive models to be used with high-dimensional sequential data. To define a VAE, we need to describe its *generative model* (i.e. the latent variable model), the *inference network* (i.e. the variational approximation), and how to learn the parameters of the VAE.

### 2.4.1 Generative model

The generative model of a VAE is given by the joint probability distribution $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})$ already introduced in (2.4). $\mathbf{z}$ is a continuous latent variable with $L$ dimensions, whose prior is typically an isotropic multivariate Gaussian with zero mean and an identity covariance matrix, i.e. $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. The likelihood $p_\theta(\mathbf{x}|\mathbf{z})$, also known as *decoder*, is typically a Gaussian distribution (for continuous data) or a Bernoulli distribution (for binary data) whose parameters are computed by passing the latent state $\mathbf{z}$ through a deep neural network. In the continuous case, we can have for example $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{v})$ where the mean $\boldsymbol{\mu}$ and the diagonal $\mathbf{v}$ of the diagonal covariance matrix are parameterized by two deep neural networks (NN) that output vectors in $\mathbb{R}^D$, with $D$ being the dimensionality of the observation $\mathbf{x}$:

$$\boldsymbol{\mu} = \mathrm{NN}_1(\mathbf{z}) \ , \qquad \log \mathbf{v} = \mathrm{NN}_2(\mathbf{z}) \ . \tag{2.10}$$

In this case, the parameters $\theta$ of the model are the weights and biases of these neural networks. The graphical model for the generative model of a VAE is shown in Figure 2.1. Notice that while we have here described a model with a single latent variable, VAEs can also be extended to be formed by multiple layers of stochastic units (Rezende et al., 2014; Sønderby et al., 2016b).

### 2.4.2 Inference network

Due to the non-linearities in the deep neural networks that parameterize $p_\theta(\mathbf{x}|\mathbf{z})$, in a VAE the exact computation of the data log-likelihood $p_\theta(\mathbf{x})$ is intractable. To perform Maximum Likelihood learning of the parameters $\theta$ of the VAE, we can however use the ELBO introduced in Section 2.3. How can we choose a variational approximation $q(\mathbf{z})$ that performs well while ensuring the scalability of the model to large data sets?

In traditional *variational inference* (Jordan et al., 1999), the variational approximation is restricted to be in a parametric family of distributions, and for each data point $\{\mathbf{x}^i\}_{i=1}^N$ we learn a different set of parameters $\phi_i$. To make this explicit in the notation, we can write the variational approximation for data point $i$ as $q_{\phi_i}(\mathbf{z}^i)$. For Gaussian variational approximations with diagonal covariance matrix, $\phi_i$ would contain for example two $L$-dimensional vectors, one for the mean and one for the diagonal of the covariance matrix. The parameters $\theta$ of the model and the parameters $\{\phi_i\}_{i=1}^N$ of the variational approximations for all data points are then learned maximizing the ELBO in (2.7). After training, if a new data point arrives, to find its variational parameters we need to optimize the ELBO again with respect to them.

This linear scaling of the parameters of the variational approximation with the number of data points can however be a problem in large data sets that may contain millions of elements and for which the variational approximation cannot be computed analytically. To deal with this issue, in a VAE we perform *amortised inference* (Gershman and Goodman, 2014). Instead of having a different set of parameters $\phi_i$ to learn for each data point, the variational parameters $\phi$ are now shared across all data points (we have therefore dropped the $i$ subscript in the notation). In a VAE in particular, we use deep neural networks that take the data point $\mathbf{x}^i$ as input, and output the mean and diagonal covariance matrix of the corresponding Gaussian variational approximation, i.e. $q_\phi(\mathbf{z}^i|\mathbf{x}^i) = \mathcal{N}(\mathbf{z}^i; \boldsymbol{\mu}_q^i, \mathbf{v}_q^i)$ with

$$\boldsymbol{\mu}_q^i = \mathrm{NN}_3(\mathbf{x}^i) \ , \qquad \log \mathbf{v}_q^i = \mathrm{NN}_4(\mathbf{x}^i) \ . \tag{2.11}$$

We therefore learn an *inference network*, also known as *encoder*, that allows us to compute the parameters of the posterior approximation given the data point. The shared variational parameters $\phi$ are now the weights and biases of the neural networks in (2.11), and the cost of learning them is amortised across all data points. Thanks to the inference network, when a previously unseen data point arrives we can immediately compute its variational approximation without the need to run an expensive optimization of the ELBO, as needed in traditional variational inference. However, the posterior approximation found with amortised inference will always be worse than the one found with the traditional approach, as the parameters of the inference network are shared across all data points. An in depth study on the impact of using inference networks in VAEs can be found in (Cremer et al., 2018), that empirically shows that the main causes of inference sub-optimality in VAEs are the approximations introduced by using amortized inference, rather than the ones introduced by restricting the family of distributions the variational approximation belongs to.

### 2.4.3 Parameter learning

In a VAE, the structure of the generative model and inference network introduced above allows fast and scalable training. As we have seen in Section 2.3, to perform Maximum Likelihood learning of the parameters $\theta$ of the model in the presence of latent variables, we can use the the ELBO $\mathcal{F}_i(\theta, q)$ introduced in (2.7). The variational approximation $q_\phi(\mathbf{z}|\mathbf{x})$ is however chosen to be in a parametric family, therefore the maximization over $q$ in (2.7) is actually a maximization over the parameters $\phi$ (we therefore use below the notation $\mathcal{F}_i(\theta, \phi)$). We can decompose the ELBO in two terms:

$$\begin{aligned}\mathcal{F}_i(\theta, \phi) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\ &= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]}_{\text{Reconstruction term}} - \underbrace{\mathrm{KL}\left[q_\phi(\mathbf{z}|\mathbf{x})||\log p_\theta(\mathbf{z})\right]}_{\text{Regularization term}} . \end{aligned} \tag{2.12}$$

The *reconstruction* term encourages the likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ and the inference network to be able to reconstruct the data accurately, maximizing therefore the auto-encoding capabilities of the VAE.[2] The second term penalizes posterior approximations that are too far from the prior, and acts therefore as a *regularization* term. As both generative and inference models are defined with neural networks, we can efficiently compute gradients of the ELBO with respect to $\theta$ and $\phi$ using the back-propagation algorithm (Rumelhart et al., 1986). Importantly, both set of parameters can be updated jointly in a single optimization step, instead of iteratively optimizing one set of parameters while keeping the other fixed as in the EM algorithm presented in Section 2.3.1. The expectation in (2.12) does not have a closed form solution, but we can obtain a low-variance differentiable unbiased estimator of the lower bound by using the *reparametrization trick* (Kingma and Welling, 2014; Rezende et al., 2014) to be able to back-propagate through the latent variable $\mathbf{z}$, and Monte Carlo integration to approximate the intractable expectation. As both $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z})$ are Gaussians, the KL term can be computed analytically (Kingma and Welling, 2014).

In Figure 2.2 (left) we visualize the 2-dimensional latent space of a small VAE trained on the first four digits (0, 1, 2 and 3) of the MNIST data set (LeCun and Cortes, 2010). The dataset

[2] In a standard auto-encoder the loss function typically used during training is in fact given by the reconstruction term in (2.12).

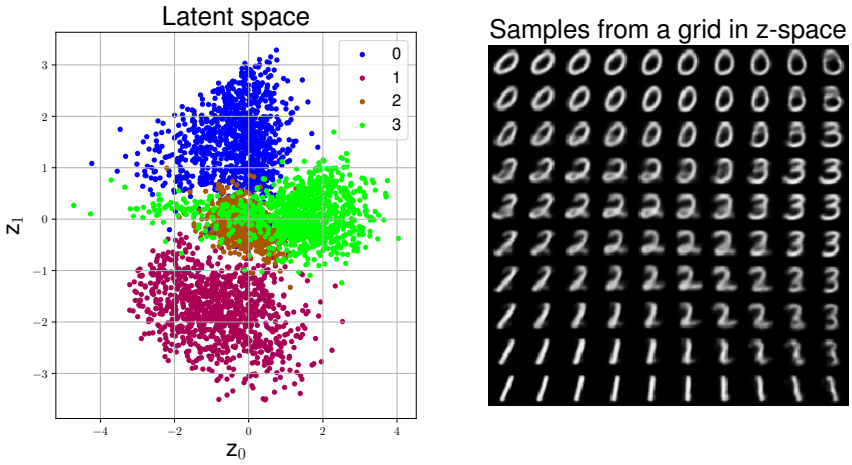



***Figure 2.2:*** *Latent representations of different MNIST digits from 0 to 3 obtained passing the images through the inference network (left). Generated samples of MNIST digits obtained passing the elements of a* $10 \times 10$ *grid in latent space through the decoder (right).*

used during training is unlabelled, i.e. it only consists of image data and does not contain any class information. Despite the model is trained in a fully unsupervised manner, the VAE learns a latent space that captures the natural clustering of the data, as by doing so it is easier to model the data distribution (a higher ELBO is achieved). Not surprisingly, this feature has led several authors to develop extensions of VAEs to the semi-supervised setting, for example to learn a classifier given only a few labelled data points (Kingma et al., 2014; Maaløe et al., 2016; Maaløe et al., 2017). Figure 2.2 also shows samples generated from a grid of points in the 2-dimensional latent space, that allow us to visualize the different latent representations each of the dimensions of $\mathbf{z}$ has learned to model the data. Looking at the last row of digits for example, it seems that the horizontal axis is responsible for modelling rotations in the digits.

## 2.5 Improving variational auto-encoders

VAEs are very flexible deep latent variable models, that can potentially model a wide range of very complex data distributions. However, this modelling power is limited by some of the assumptions and approximations that are necessary to be able to define a scalable architecture.

As we have seen in Section, 2.3, instead of maximizing the intractable data log-likelihood $\log p_\theta(\mathbf{x})$ we are maximizing the ELBO, relying on the fact that as the data log-likelihood is lower-bounded by the ELBO, an high ELBO implies a high value for $\log p_\theta(\mathbf{x})$. However, we have no guarantees on how tight the bound is, or whether the parameters that maximize the ELBO are also a maximum of the data log-likelihood. For the bound to be tight, the variational approximation needs to be as close as possible to the true posterior distribution, but for practical and computational reasons we are often forced to use a simple Gaussian approximation that cannot properly fit the complex multimodal posterior distributions of deep latent variable models. Other challenges come from the fact that we are doing MLE and posterior inference jointly: as we are maximizing over both the parameters $\theta$ of the generative model and $\phi$ of the inference network, it is possible to learn a suboptimal generative model whose parameters depend on the choice of the posterior

approximation (instead of being independent from it as if we could maximize over $\log p_\theta(\mathbf{x})$ in an exact way). $\theta$ defines in fact the shape of the true posterior distribution (2.3), and the optimal $\theta$ that maximizes the ELBO will be one such that the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ can be better approximated by the parametric family chosen for the variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$. In this way we avoid the heavy penalizations in the ELBO for posterior samples that do not explain the observations well enough.

We will now review different extensions to the basic model presented in Section 2.4, that were proposed in the literature to solve some of these issues.

### 2.5.1 Tightening the ELBO

We can use importance sampling to define a new lower bound for $\log p_\theta(\mathbf{x})$ that is tighter than the ELBO. The resulting model is called *importance weighted auto-encoder* (IWAE) (Burda et al., 2015). To define this new objective function, we sample $K$ latent variables $\{\mathbf{z}^{(k)}\}_{k=1}^K$ independently from the same inference network $q_\phi(\mathbf{z}|\mathbf{x})$ used in Section 2.4.2, and we compute the $K$ sample importance-weighted estimate of the log-likelihood

$$\mathcal{F}_i^K(\theta,\phi) = \mathbb{E}_{\mathbf{z}^{(1)},\dots,\mathbf{z}^{(K)}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(\mathbf{x},\mathbf{z}^{(k)})}{q_\phi(\mathbf{z}^{(k)}|\mathbf{x})}\right] . \tag{2.13}$$

Importantly, for $K=1$ the bound in (2.13) reduces to the ELBO introduced in (2.12), and using more samples can only improve the tightness of the bound, i.e. for all $K$ we have

$$\log p_\theta(\mathbf{x}) \geq \mathcal{F}_i^{K+1}(\theta,\phi) \geq \mathcal{F}_i^K(\theta,\phi) .$$

Using more computational power we can therefore to improve the modelling performances. The use of multiple posterior samples in the computation of the bound allows us to learn generative model with a complex posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ that would not be approximated well by a single sample from the simple variational approximation of a VAE.

The work from Li and Turner, (2016) has extended traditional variational inference by defining objective functions based on the *Rényi's $\alpha$-divergence*, a broad family of divergences that extends both the objectives in (2.12) and in (2.13), and that allows to define lower and upper bounds to the data log-likelihood.

### 2.5.2 Defining flexible posterior approximations

Ideally, we would like to have a very flexible inference network that can approximate very complex posterior distributions. However, to make inference and parameter learning scalable, we often restrict the variational approximation to a simple parametric distribution, e.g. a factorized Gaussian. Can we build more flexible inference networks without increasing too much the computational costs? As we will see below, there are different ways to achieve this.

**Normalizing flows.** We can form a complex probability density $q_R(\mathbf{z}_R)$ by starting from a simple density $q_0(\mathbf{z}_0)$ and constructing a chain of $R$ invertible parametric transformations $f_r$, known as *normalizing flows*, that expand and contract the initial density: $\mathbf{z}_R = f_R(f_{R-1}(\dots f_1(\mathbf{z}_0)))$. The resulting probability density over the variable $\mathbf{z}_R$ can be computed by repeatedly applying the rule for change of variables, giving

$$\log q_R(\mathbf{z}_R) = \log q_0(\mathbf{z}_0) - \sum_{r=1}^{R} \log \left| \det \frac{\partial f_r}{\partial \mathbf{z}_{r-1}} \right| .$$

In a VAE in particular, we can use this density to parameterize the approximate posterior distribution, i.e. defining $q_\phi(\mathbf{z}|\mathbf{x}) \triangleq q_R(\mathbf{z}_R)$, choosing as initial density $q_0(\mathbf{z}_0)$ the same inference network introduced in Section 2.4.2 (Rezende and Mohamed, 2015). We can use amortized inference to learn the parameters of the flow, by making them dependent on the data point $\mathbf{x}$ through a deep neural network. The scalability of this approach can be ensured by choosing relatively simple transformations that have an efficient mechanism for computing the determinant of the Jacobian $\frac{\partial f_r}{\mathbf{z}_{r-1}}$, as needed when evaluating $q_R(\mathbf{z}_R)$ in the ELBO. Some possible choices are planar and radial flows (Rezende and Mohamed, 2015), inverse auto-regressive flows (Kingma et al., 2016), Hamiltonian flows (Salimans et al., 2015) and masked auto-regressive flows (Papamakarios et al., 2017).

**Auxiliary variables.** An alternative way to increase the flexibility of the variational approximation is by introducing an auxiliary latent variable $\mathbf{a}$ that parameterizes the joint variational approximation in the extended space as $q_\phi(\mathbf{z}, \mathbf{a}|\mathbf{x}) = q_\phi(\mathbf{z}|\mathbf{a}, \mathbf{x})q_\phi(\mathbf{a}|\mathbf{x})$, while keeping the generative model unchanged (Agakov and Barber, 2004; Ranganath et al., 2015; Maaløe et al., 2016). Both distributions on the right hand side of the previous equation are parameterized by deep networks. The variational approximation over $\mathbf{z}$ only is then obtained by integrating out the auxiliary variables,

$$q_\phi(\mathbf{z}|\mathbf{x}) = \int q_\phi(\mathbf{z}|\mathbf{a}, \mathbf{x})q_\phi(\mathbf{a}|\mathbf{x})\mathrm{d}\mathbf{a} .$$

This integral is however intractable, therefore we cannot evaluate the ELBO in (2.12). As shown in (Ranganath et al., 2015; Maaløe et al., 2016), this can be solved by maximizing a tractable lower bound to the ELBO instead of the ELBO itself.

**Variational Boosting.** This method can be used to iteratively refine the variational approximation by incorporating additional covariance structure and by introducing new components to form a mixture (Miller et al., 2017).

## 2.6 Summary and discussion

In this chapter we have seen that LVMs can be used to model complex high-dimensional data distributions (Section 2.1). DLVMs use deep neural networks to parameterize the probability distributions that define a LVM. While this allows to define very flexible models, it makes exact inference and parameter learning intractable. We have then shown in Sections 2.2 and 2.3 how they can be approximated with variational methods. In Section 2.4 we have introduced the VAE

as a basic example of DLVM, and showed different ways to improve its performances in Section 2.5.

So far we have used LVMs to model *static* data, i.e. data with no temporal dependencies such as the images of handwritten digits used in the VAE example in Section 2.4.3. If we think about the images that form the frames of a video however, we know that there will be some temporal correlations among them, i.e. we are dealing with *dynamic* data. In this case, we are interested in modelling a sequence of observations (the whole video) instead of a single observation (one frame). Consider a data set of $N$ sequences of high-dimensional observations $\mathbf{x}^i_{1:T} = [\mathbf{x}^i_1, \dots, \mathbf{x}^i_T]$, $i = 1, \dots, N$, from which we want to learn a model for the joint probability distribution over a sequence $p_\theta(\mathbf{x}_{1:T})$. For simplicity in the exposition, we make the assumption that the number of time steps $T$ of all sequences is fixed, but the generalization of the ideas presented below to sequences with variable length $T_i$ is straightforward. We could in principle treat the dynamic data as static by assuming the factorization $p_\theta(\mathbf{x}_{1:T}) = \prod_{t=1}^T p(\mathbf{x}_t)$, and fit a LVM (e.g. a VAE) to the data set of $NT$ data points treated as if they were independent. While this can be a good model for a single observation, it does not capture the temporal nature of the data. To do it, we need to introduce some dependencies in the latent states $\mathbf{z}^i_{1:T}$ corresponding to all observations of a sequence, instead of considering $p_\theta(\mathbf{z}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t)$ as implicitly done when treating the data used to train the LVM as static. In other words, we want to introduce some temporal structure in the prior $p_\theta(\mathbf{z}_{1:T})$ over the latent variables of a sequence. In the next chapter we will assume that the latent variables form a chain, so that at each time step $t$, the variable $\mathbf{z}_t$ directly depends on $\mathbf{z}_{t-1}$. Latent variable models for sequential data that add this direct dependency among latent variables have been studied - although in different contexts - for more than half a century, since the seminal paper by (Kalman, 1960). This class of models is commonly referred to as *state-space models* (where the term "state" is used to denote what we called in this chapter "latent variables") and will be the focus of the next chapter.

Chapter 3

# State-space models

*State-space models* (SSM) provide a general and flexible methodology for sequential data modelling. They were first introduced in the 1960s, with the seminal work of Kalman, (1960), and were soon used in the Apollo Project to estimate the trajectory of the spaceships that were bringing men to the moon (Mcgee and Schmidt, 1985). Since then, they have become a standard tool for time series analysis in many areas well beyond aerospace engineering. In the machine learning community in particular, they are used as generative models for sequential data, for predictive modelling, state inference and representation learning. An excellent treatment of state-space modelling for time series analysis can be found in the in book by Durbin and Koopman, (2012).

## 3.1 Definitions

We are given a sequence of $T$ observation $\mathbf{x}_{1:T} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$, that possibly depend on some inputs $\mathbf{u}_{1:T} = [\mathbf{u}_1, \ldots, \mathbf{u}_T]$, and we are interested in modelling the distribution $p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T})$. This is a very general formulation, that can be applied in a wide variety of applications. We may want to model for example how the movements of the steering wheel and of the brake/throttle pedals (the inputs/controls to the model) change the position of a car (the observations/outputs). Using $\mathbf{u}_t = \mathbf{x}_{t-1}$ it is also possible to define autoregressive models, as typically used in the deep learning community when building generative models for text, videos or speech.

In a SSM we introduce at each time step a *state* variable $\mathbf{z}_t$ that summarizes all the information coming from the past and determines the present and future evolution of the system. SSMs can then be seen as a temporal extension to the latent variable model introduced in Section 2.1, in which the prior over the latent variables $\mathbf{z}_t$ at each time step varies over time as it depends on the previous state $\mathbf{z}_{t-1}$ and possibly some inputs $\mathbf{u}_t$ to the model. We assume that the joint

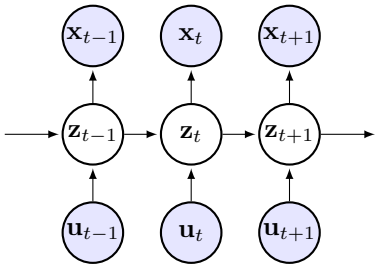



***Figure 3.1:*** *A graphical representation of a state-space model.*

distributions of observations and states given the inputs factorizes as

$$\begin{aligned} p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}) &= p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) p_\theta(\mathbf{z}_{1:T}|\mathbf{u}_{1:T}) \\ &= \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{z}_t) \cdot p_\theta(\mathbf{z}_1) \prod_{t=2}^{T} p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) \end{aligned} \tag{3.1}$$

A graphical representation of the distribution in (3.1) can be found in Figure 3.1. The *emission distribution* $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$ specifies how the observation $\mathbf{x}_t$ depends on the latent state $\mathbf{z}_t$, and can therefore be seen as the likelihood in a LVM. $p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)$ is called *transition distribution*, and represents the prior distribution for the state at each time step given the previous state and the current input to the model. This distribution fully determines the temporal evolution of the system. The states of the SSM form a Markov chain, that captures the temporal correlations and long-term dependencies between observations at different time steps. Using the *d-separation* properties (Geiger et al., 1990) of the graphical model in Figure 3.1, we can see that this Markovian structure leads to some interesting conditional independence properties that are implicitly assumed in a SSM:

$p_\theta(\mathbf{x}_t|\mathbf{z}_{1:t}, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) = p_\theta(\mathbf{x}_t|\mathbf{z}_t)$
: This property implies that given the present state $\mathbf{z}_t$ the observation at time $t$ does not depend on the past states, inputs and outputs of the model. As in a LVM, the observation $\mathbf{x}_t$ is then fully determined by the latent state $\mathbf{z}_t$.

$p_\theta(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) = p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)$
: Conditioned on $\mathbf{z}_{t-1}$, the current state $\mathbf{z}_t$ does not depend on the previous states $\mathbf{z}_{1:t-2}$, nor the past inputs or outputs. $\mathbf{z}_{t-1}$ then captures all the relevant information on the past.

$p_\theta(\mathbf{z}_t|\mathbf{z}_{t+1:T}, \mathbf{x}_{t+1:T}, \mathbf{u}_{t+1:T}) = p_\theta(\mathbf{z}_t|\mathbf{z}_{t+1})$
: Given the next state $\mathbf{z}_{t+1}$, $\mathbf{z}_t$ does not depend on the future states, inputs and outputs, i.e. $\mathbf{z}_{t+1}$ captures all the relevant information on the future.

As we will see in the following, these conditional independence relationships are responsible for many of the nice properties of SSMs.

Similarly to the LVM in Section 2.1, the marginal distribution over the observations can be obtained by integrating out the states in (3.1), i.e.

$$p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T}) = \int p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}) \mathrm{d}\mathbf{z}_{1:T} \ . \tag{3.2}$$

Here we have assumed that $\mathbf{z}_{1:T}$ are continuous variables, but the same ideas apply to the discrete case by replacing the integral with a summation. Using Bayes' rule we obtain the posterior distribution of the states given the data:

$$p_\theta(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}) = \frac{p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T})p_\theta(\mathbf{z}_{1:T}|\mathbf{u}_{1:T})}{p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T})} \ . \tag{3.3}$$

Exact posterior inference is analytically tractable for two classes of SSMs, namely Linear Gaussian SSMs and Hidden Markov Models. In all other cases we will need to resort to approximate inference, as discussed more in detail in Section 3.5.1.

In some cases we know the exact form of the emission and transition distributions, and we are only interested in inferring the latent states for a given sequence, as shown in the ball tracking example that will be presented in Section 3.4. SSMs can however be also used as black-box methods for sequential data modelling, in which case the emission and transition distribution will have a flexible structure that can be learned from the data, see (Fraccaro et al., 2016c, Chapter 5) for an example of this approach. Finally, in other cases we can use prior information on the task at hand to define a specific parametric form for the emission and transition distribution that helps the model to learn meaningful and interpretable latent representations. This approach lies halfway between having a known or a black-box model, and will be used in (Fraccaro et al., 2017, Chapter 6) and in (Fraccaro et al., 2018, Chapter 7).

## 3.2 Classes of state-space models

Depending on the exact form of the emission and transition distributions we can define several classes of state-space models, that will be briefly presented in the following.

**Linear Gaussian state-space models.** The simplest class of SSMs is that of Linear Gaussian state-space models (LGSSM), first introduced in (Kalman, 1960). We will present them in detail in Section 3.4. As suggested by the name, both transition and emission distributions are Gaussians, and all relationships between states and observations are linear. This makes posterior inference for this model analytically tractable. Despite its simplicity, the LGSSM can be seen as a generalization of many classical models used in time series analysis. As shown in (Durbin and Koopman, 2012), the widely used autoregressive integrated moving average (ARIMA) model can be expressed in state-space form. LGSSM can also model in a unified framework trends, seasonal components, explanatory variables and interventions.

**Hidden Markov models.** A Hidden Markov model (HMM) is a SSM with discrete latent states (Rabiner, 1990). As for the LGSSM, for a HMM we can perform exact posterior inference. Notice that HMMs were developed in parallel to LGSSM, therefore some authors prefer to reserve the term "state-space models" for models with continuous states, and use the term "hidden Markov models" when dealing with discrete states. Here however we prefer to consider HMMs as SSMs, as they satisfy all the assumptions we made in Section 3.1 (e.g. the factorization of the joint distribution in (3.1)).

**Non-Linear non-Gaussian state-space models.** The linear-Gaussian assumptions of a LGSSM are often too restrictive for many applications. If we relax them however we introduce an

additional challenge, since inference becomes intractable and we need to resort to approximate methods. See Section 3.5 for details. As we will see in Section 4.3, a flexible class of non-linear state space models is given by deep state-space models (DSSM). In a DSSM, the transition and emission distributions are parameterized with deep neural networks, and efficient training can be achieved with amortized variational inference, computing the required gradients with the back-propagation algorithm. For small data sets it is also common to model the transitions and emissions with Gaussian processes, see (McHutchon, 2014) for an overview on the topic.

**Hybrid architectures.** Different authors have combined in a single architecture different classes of SSMs. Murphy, (1998) and Ghahramani and Hinton, (2000) introduce for example Switching Kalman filters, that use the discrete states of a HMM to select different possible regimes for the continuous variables of a LGSSM.

**SSMs in continuous time.** In Section 3.1 we have tacitly assumed that the data is observed at equally-spaced discrete time steps. SSMs can however be used also to model continuous time systems, in which the state is used to represent the dynamics of higher-order linear systems as a first order differential equation.

## 3.3 Posterior inference in the sequential setting

In (3.3) we have expressed the posterior distribution of the latent states given the whole sequence. However, if we take into account the temporal structure of the data, there are also other types of inference we can be interested in.

To illustrate this, consider the simplified speech recognition example of trying to understand what a friend is saying in a very noisy bar. The observation $\mathbf{x}_t$ represents the noisy speech waveform at each time step, while $\mathbf{z}_t$ is the discrete variable that represents the corresponding word pronounced by the friend.[1] In this example there are no inputs $\mathbf{u}_t$ to the model, and we will therefore remove them from the equations. At any point in time, we want of course to infer the word that the friend is saying, i.e. compute the posterior distribution $p_\theta(\mathbf{z}_t|\mathbf{x}_t)$. As the bar is noisy however, we may not be sure of which word was pronounced. At time $t$ we also know what the friend said in $\mathbf{x}_{1:t-1}$, and we can therefore condition even on the past observations, i.e. compute $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t})$ instead. The knowledge about what the friend was talking about at previous time steps can provide some context and help us better infer $\mathbf{z}_t$. We call this task *filtering*, as it reduces the noise compared to only using the present observation $\mathbf{x}_t$ during inference. Despite this, due to the noise in the bar we may still be unsure on the inferred word. In this case we can hope that while we keep listening to the friend, there will be one clue that will clarify the inferred state $\mathbf{z}_t$. In this case we are therefore also using knowledge on the future during inference, i.e. we are considering the *smoothed* posterior $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:T})$. Finally, we may also want to predict what the friend will say in the future given what was said until now, i.e. compute $p_\theta(\mathbf{z}_{t+k}|\mathbf{x}_{1:t})$.

[1] Before the advent of deep learning, HMMs have been widely used for speech recognition (Gales and Young, 2008). Instead of using the raw waveform as observation, these systems typically use a frequency-domain representation of it (e.g. cepstral coefficients). Instead of modelling words in the latent states, it is more common to model phonemes. For clarity in the exposition, this is not discussed in this simplified example.

We now provide a more general description of these inference tasks:

**Filtering.** We want to compute the filtered posterior distribution of the state $\mathbf{z}_t$ given present and past input and output information, i.e. $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$. This task is particularly interesting in an online setting, as it allows to compute the state estimate as the data comes in.

**Smoothing.** When doing smoothing, we compute the posterior $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:T}, \mathbf{u}_{1:T})$, conditioned not only on the past and present information, but also on future one. Since the smoothed posterior requires the knowledge of the whole sequence, it can be computed only offline. A trade-off between filtering and smoothing is *fixed-lag smoothing*, where we compute the smoothed posterior only conditioning on data up to $k$ time steps in the future (and not on the whole sequence), i.e. we compute $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t+k}, \mathbf{u}_{1:t+k})$. Fixed-lag smoothing can be used to further improve state estimation in an online setting, whenever a delay of $k$ time steps in admissible.

**Prediction.** We can also be interested in predicting the state of the system $k$ steps in the future given only past information, i.e. computing $p_\theta(\mathbf{z}_{t+k}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t+k})$ (notice that if the inputs $\mathbf{u}_t$ are present, they need to be known up to time $t+k$).

## 3.4 Linear Gaussian state-space models

We now discuss in detail the *linear Gaussian state-space model* (Kalman, 1960), that was briefly introduced in Section 3.2. This model is also known as *Linear Dynamical System*, or more informally as *Kalman Filter*. A LGSSM is typically written in terms of two equations, that specify the relationship between the latent states at consecutive time steps and the observations:

$$\mathbf{z}_t = \mathbf{A}_t\mathbf{z}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \boldsymbol{\varepsilon}_t \tag{3.4}$$

$$\mathbf{x}_t = \mathbf{C}_t\mathbf{z}_t + \boldsymbol{\delta}_t \tag{3.5}$$

The *transition model* (3.4) describes how to compute the state $\mathbf{z}_t$ at each time step given the previous state $\mathbf{z}_{t-1}$ and the current input $\mathbf{u}_t$. $\mathbf{A}_t$ and $\mathbf{B}_t$ are the *transition* and *control* matrices respectively, and define a linear relationship between the variables. The transitions are perturbed with a Gaussian *process/transition noise* $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(\boldsymbol{\varepsilon}_t; \mathbf{0}, \mathbf{Q}_t)$, where $\mathbf{Q}_t$ is called *transition covariance*. In a LGSSM we do not observe the state, but only a linearly transformed version of it with additive Gaussian noise, as specified by the *emission model* (3.5). The emission model can be seen as a linear regression model with time-varying inputs $\mathbf{z}_t$. In (3.5) $\mathbf{C}_t$ is called *emission* matrix, and the *measurement/observation noise* $\boldsymbol{\delta}_t$ is a Gaussian random variable, i.e. $\boldsymbol{\delta}_t \sim \mathcal{N}(\boldsymbol{\delta}_t; \mathbf{0}, \mathbf{R}_t)$ with $\mathbf{R}_t$ being the *observation covariance*. Both transition noise and observation noise are assumed to be independent across time steps. The matrices that define the LGSSM can be time-varying (as indicated by the subscript $t$) and even depend on the past information. However, for simplicity often the matrices are kept fixed over time, in which case the model is defined as *stationary* (we will see an example of this at the end of this section).

The transition and emission distributions of a LGSSM can be easily obtained from (3.4) and (3.5) using the linear transformation properties of Gaussian random variables:

$$p_{\theta_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t; \mathbf{A}_t\mathbf{z}_{t-1} + \mathbf{B}_t\mathbf{u}_t, \mathbf{Q}_t) \tag{3.6}$$

$$p_{\theta_t}(\mathbf{x}_t|\mathbf{z}_t) = \mathcal{N}(\mathbf{x}_t; \mathbf{C}_t\mathbf{z}_t, \mathbf{R}_t)\ , \tag{3.7}$$

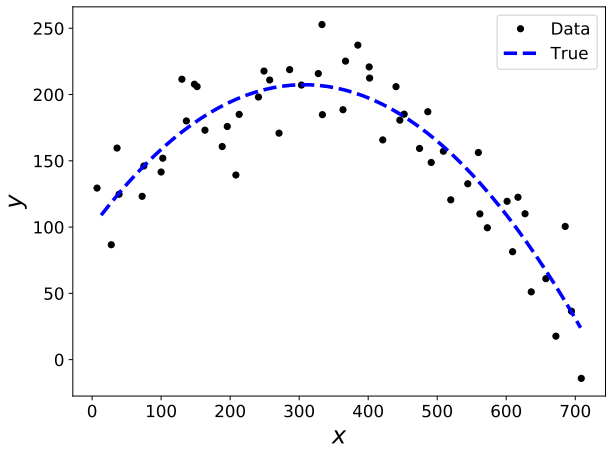



***Figure 3.2:*** *Ball tracking example, $(x, y)$ plane.*

where we have defined $\theta_t = [\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t, \mathbf{Q}_t, \mathbf{R}_t]$. We assume that the state at the first time step is a Gaussian random variable, i.e. $p(\mathbf{z}_1) = \mathcal{N}(\mathbf{z}_1; \boldsymbol{\mu}_{1|0}, \boldsymbol{\Sigma}_{1|0})$. The joint distribution in (3.1) then becomes:

$$\begin{aligned} p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}) &= p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) p_\theta(\mathbf{z}_{1:T}|\mathbf{u}_{1:T}) \\ &= \prod_{t=1}^{T} p_{\theta_t}(\mathbf{x}_t|\mathbf{z}_t) \cdot p(\mathbf{z}_1) \prod_{t=2}^{T} p_{\theta_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) \end{aligned} \tag{3.8}$$

where $\theta = [\boldsymbol{\mu}_{1|0}, \boldsymbol{\Sigma}_{1|0}, \theta_1, .., \theta_T]$.

One of the major reasons for the widespread usage of LGSSMs is that posterior inference can be done in an exact way, as we will discuss in Section 3.4.1. In Sections 3.4.2 and 3.4.3 we will then see that the LGSSM provides simple methods to perform missing data imputation and parameter learning. These properties will be exploited in the model of Fraccaro et al., (2017, Chapter 6). Throughout this section we will use a simple example, introduced below, to showcase the capabilities of LGSSMs.

**Ball tracking example.** LGSSMs are widely used for real-time object tracking, e.g. to estimate the position of satellites or in GPS systems. To see why LGSSM are suitable for such applications, we can look at a simplified example of tracking a ball thrown in vacuum from noisy measurements of its position. The ball is subject to gravity, and we can measure with a noisy sensor its $(x, y)$ position but not its velocity or acceleration. We assume that the data is sampled every $\Delta = 0.2$ seconds. Figure 3.2 shows the true trajectory of the ball in the $(x, y)$ plane (dashed blue line) and the noisy measurements (black dots). As expected, due to the force of gravity acting on the ball, the trajectory forms a parabola.

To define the parameters of a suitable LGSSM we can exploit our knowledge of the physics of the moving ball. The discretized system dynamics can be modelled using Newton's equations of

motion as follows

$$\begin{cases} x_t &= x_{t-1} + \dot{x}_{t-1}\Delta \\ \dot{x}_t &= \dot{x}_{t-1} \\ y_t &= y_{t-1} + \dot{y}_{t-1}\Delta - \frac{1}{2}g\Delta^2 \\ \dot{y}_t &= \dot{y}_{t-1} - g\Delta \end{cases} \tag{3.9}$$

where $\dot{x}_t$ and $\dot{y}_t$ represent the velocities on the $x$ (horizontal) and $y$ (vertical) axes respectively, and $g = 9.81\frac{m}{s^2}$ is the acceleration due to gravity. From these equations we can see that the ball is following a linear motion with constant velocity on the $x$ axis, whereas on the $y$ axis there is an acceleration due to the force of gravity. The state of the system at each time step is given by the positions $(x_t, y_t)$ and velocities $(\dot{x}_t, \dot{y}_t)$ of the ball. The true state is however unknown, since our sensor can only measure the noisy position of the ball. We then need to be able to estimate the true state from these noisy observations, task for which a stationary LGSSM is the ideal candidate.

We define the emission matrix and the state of the systems as

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad \mathbf{z}_t = \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{bmatrix},$$

so that the noisy observations $\mathbf{x}_t$ can be modelled with (3.5):

$$\mathbf{x}_t = \mathbf{C}\mathbf{z}_t + \boldsymbol{\delta}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \boldsymbol{\delta}_t \,. \tag{3.10}$$

The emission matrix derived from (3.9) is time-independent, and we have therefore removed the subscript $t$ that was used previously in this section. We further define the transition matrix, control matrix and control inputs as:

$$\mathbf{A} = \begin{bmatrix} 1 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2}\Delta^2 & 0 \\ 0 & 0 & 0 & -\Delta \end{bmatrix}, \qquad \mathbf{u}_t = \begin{bmatrix} 0 \\ 0 \\ g \\ g \end{bmatrix},$$

so that the noiseless transition equation $\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1} + \mathbf{B}\mathbf{u}_t$ will return exactly the system dynamics of (3.9). Notice in particular that the gravity, an external force applied to the ball, can be modelled using a fixed control input $\mathbf{u}_t$ at each time step. We could have otherwise added it as an extra fixed term in the state vector.

As Newton's equations of motions give us the exact dynamics of a ball moving in the vacuum, we set the transition noise covariance $\mathbf{Q}$ to zero. We then set $R = 3\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. We assume that we do not know anything about the initial state $\mathbf{z}_1$, and we set its parameters to $\boldsymbol{\mu}_{1|0} = \mathbf{0}$ and $\boldsymbol{\Sigma}_{1|0} = 2\mathbf{I}$. This is a relatively concentrated Gaussian around the origin that assigns a low probability to the true initial state. This is done to show in the next sections that exact inference works well despite this and how we can learn this vector.

### 3.4.1 Posterior inference

For LGSSMs the filtered and smoothed posteriors at each time step can be computed analytically using the Kalman filtering and smoothing algorithms. The proofs of the algorithms rely heavily both on the Markovian structure of SSMs and on the linear-Gaussian assumptions, that imply that the joint distribution over all variables, as well as all marginals and conditionals will be Gaussians. In the following we will present the main intuitions behind this algorithms and a sketch of the proofs. A detailed derivation can be found in (Murphy, 2012).

#### Filtering

In this section we present a filtering routine for the LGSSM known as *Kalman filtering*. The Kalman filtering algorithm (Kalman, 1960) recursively computes the marginal posterior distribution $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$ at each time step given the one at the previous time step, $p_\theta(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1})$. Using Bayes' rule and the conditional independence properties of the model, we can rewrite this posterior distribution as follows:

$$\begin{aligned} p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) &= p_\theta(\mathbf{z}_t|\mathbf{x}_t, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) \\ &= \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t})}{p_\theta(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t})} \\ &= \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t) p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t})}{p_\theta(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t})} \,. \end{aligned} \tag{3.11}$$

The first term of the numerator is the emission distribution $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$. The second one, $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t})$, can be seen as the *predictive prior* for $\mathbf{z}_t$, i.e. our "best guess" on the distribution of the state at time $t$ given the output information up to time $t-1$. At each time step, the Kalman filter first performs a *prediction step* that computes the predictive prior given the filtered posterior at the previous time step, followed by a *measurement step* in which this distribution is updated using the information coming from the current observation that is carried by the emission distribution. As typically done when working with Gaussian distributions, for the computations we can only focus on the unnormalized distribution at the numerator of (3.11), given by the product of two Gaussians.

**Prediction step.** We assume that $p_\theta(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1}) = \mathcal{N}(\mathbf{z}_{t-1}; \boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$, the filtered posterior at the previous time step, is known. Given this distribution and the transition distribution of the LGSSM, the predictive prior can be computed as[2]

$$\begin{aligned} p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) &= \int p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) p_\theta(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) \mathrm{d}\mathbf{z}_{t-1} \\ &= \int p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) p_\theta(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1}) \mathrm{d}\mathbf{z}_{t-1} \\ &= \int \mathcal{N}(\mathbf{z}_t; \mathbf{A}_t\mathbf{z}_{t-1} + \mathbf{B}_t\mathbf{u}_t, \mathbf{Q}_t) \mathcal{N}(\mathbf{z}_{t-1}; \boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}) \mathrm{d}\mathbf{z}_{t-1} \\ &= \mathcal{N}(\mathbf{z}_t; \underbrace{\mathbf{A}_t\boldsymbol{\mu}_{t-1} + \mathbf{B}_t\mathbf{u}_t}_{\boldsymbol{\mu}_{t|t-1}}, \underbrace{\mathbf{A}_t\boldsymbol{\Sigma}_{t-1}\mathbf{A}_t^T + \mathbf{Q}_t}_{\boldsymbol{\Sigma}_{t|t-1}}) \,, \end{aligned}$$

[2] In $\mathbf{A}_t^T$, the superscript $T$ denotes the transpose operator; the sequence length, also denoted with $T$, will instead always appear as a subscript.

where we have used the conditional independence properties of the model and the well known rules on the marginalization of conditional Gaussians (Bishop, 2006). Notice in particular that to compute the mean $\boldsymbol{\mu}_{t|t-1}$ of this distribution we can simply apply the transition equations to the mean $\boldsymbol{\mu}_{t-1}$ of the previous filtered posterior. The predictive prior at the first time step is given by the initial distribution $p(\mathbf{z}_1) = \mathcal{N}(\mathbf{z}_1; \boldsymbol{\mu}_{1|0}, \boldsymbol{\Sigma}_{1|0})$.

**Measurement step.** In (3.11) we have seen that

$$\begin{aligned} p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) &\propto p_\theta(\mathbf{x}_t|\mathbf{z}_t) p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) \\ &\propto \mathcal{N}(\mathbf{x}_t; \mathbf{C}_t\mathbf{z}_t, \mathbf{R}_t)\mathcal{N}(\mathbf{z}_t; \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \ . \end{aligned}$$

From this, the mean $\boldsymbol{\mu}_t$ and the covariance matrix $\boldsymbol{\Sigma}_t$ of the filtered posterior at time $t$ can be computed using the formulas for Bayes' rule for Gaussian distributions of this form and applying the matrix inversion lemmas (Murphy, 2012). This gives

$$\begin{aligned} \boldsymbol{\mu}_t &= \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\mathbf{x}_t - \mathbf{C}_t\boldsymbol{\mu}_{t|t-1}) \\ \boldsymbol{\Sigma}_t &= (\mathbf{I} - \mathbf{K}_t\mathbf{C}_t)\boldsymbol{\Sigma}_{t|t-1} \ , \end{aligned}$$

where we have defined the *Kalman gain matrix* $\mathbf{K}_t$ as

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T(\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T + \mathbf{R}_t)^{-1} \ .$$

To understand the algorithm, it is instructive to look in particular at the update equations for the mean. The posterior mean $\boldsymbol{\mu}_t$ is obtained shifting the mean of the predictive prior by a factor proportional to the *residual* $\mathbf{r}_t = \mathbf{x}_t - \mathbf{C}_t\boldsymbol{\mu}_{t|t-1}$, that is the difference between the true observation and the predicted one. As the Kalman gain $\mathbf{K}_t$ grows when the observation covariance $\mathbf{R}_t$ becomes smaller, the Kalman filter will give more importance to the true observation when the emission noise is smaller. On the other hand, if the covariance of the predictive prior $\boldsymbol{\Sigma}_{t|t-1}$ is small (i.e. the model is quite sure of its estimation) $\mathbf{K}_t$ decreases, therefore reducing the contribution of the residual.

It is important to notice that the prediction and measurement steps of the Kalman filter only depend on the filtered posterior at the previous time step and the current input $\mathbf{u}_t$ and output $\mathbf{x}_t$. This implies that there is no need to reprocess the whole sequences of $\mathbf{x}_{1:t-1}$ and $\mathbf{u}_{1:t-1}$ at each inference step, making this iterative algorithm efficient and suitable for the online setting. The computational complexity of the Kalman filter scales cubically in the output dimensionality (due to the matrix inversion in $\mathbf{K}_t$) and quadratically in the state size (due to the matrix multiplication for $\boldsymbol{\Sigma}_t$), that make it not efficient for very high-dimensional problems (Murphy, 2012). To avoid issues with singular matrices, some more numerically stable versions this algorithm have been developed. Among these, we find for example the *square-root filter*, that works with Cholesky decompositions of covariance matrices, and the *information filter*, that updates the natural parameters of the Gaussians instead of the moments.

Given the output of the Kalman filter, it is also possible to compute the marginal likelihood $p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T})$ as

$$\log p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T}) = \sum_{t=1}^{T} \log p_\theta(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) = \mathcal{N}(\mathbf{x}_t; \mathbf{C}_t\boldsymbol{\mu}_{t|t-1}, \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T + \mathbf{R}_t) \ . \tag{3.12}$$

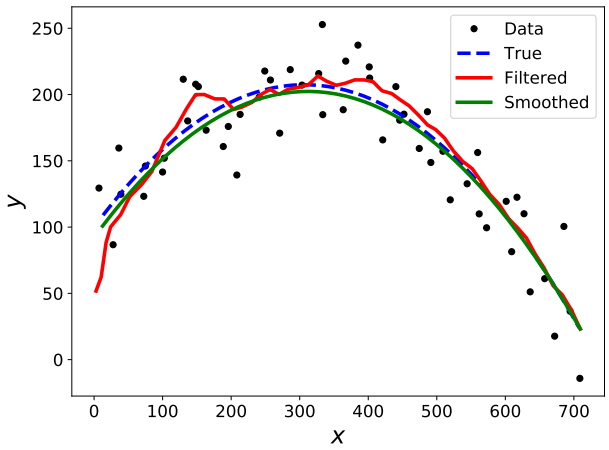



***Figure 3.3:*** *Inference in the ball tracking example.*

### Smoothing

The Kalman filter processes the whole sequence recursively forwards in time. From its output, we can easily compute also the smoothed posterior distribution with the *Rauch-Tung-Striebel smoother* (Rauch et al., 1965), also known as *Kalman smoother*. After running a forward pass with the Kalman filter, the algorithm does a backward recursion during which it combines information coming from the future observations with the quantities computed during the forward pass.

We initialize the Kalman smoother using the last step of the Kalman filter, i.e. $p_\theta(\mathbf{z}_T|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}) = \mathcal{N}(\mathbf{z}_T; \boldsymbol{\mu}_T, \boldsymbol{\Sigma}_T)$. To compute the smoothed distribution at time $t$, we then process the sequence backwards in time, combining the smoothed distribution at time $t+1$, denoted as $p_\theta(\mathbf{z}_{t+1}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}) = \mathcal{N}(\mathbf{z}_{t+1}; \boldsymbol{\mu}_{t+1|T}, \boldsymbol{\Sigma}_{t+1|T})$, with the parameters obtained during prediction step and the measurement step of the Kalman filter. We then obtain $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}) = \mathcal{N}(\mathbf{z}_t; \boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T})$, with

$$\begin{aligned}\boldsymbol{\mu}_{t|T} &= \boldsymbol{\mu}_t + \mathbf{J}_t(\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \\ \boldsymbol{\Sigma}_{t|T} &= \boldsymbol{\Sigma}_t + \mathbf{J}_t(\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t})\mathbf{J}_t^T \ ,\end{aligned}$$

where $\mathbf{J}_t = \boldsymbol{\Sigma}_t \mathbf{A}_{t+1}^T \boldsymbol{\Sigma}_{t+1|t}^{-1}$.

A derivation of the algorithm, that exploits the fact that $\mathbf{z}_t$ does not depend on $\mathbf{x}_{t+1:T}$ and $\mathbf{u}_{t+1:T}$ if we condition on $\mathbf{z}_{t+1}$, can be found in (Murphy, 2012).

**Ball tracking example.** We run the Kalman filter and smoother on the ball tracking example presented in Section 3.4. In Figure 3.3 we plot the true trajectory, together with the first and third components of the estimated state vector (i.e. the estimated $x$ and $y$ coordinates). We first notice that the filtered posterior distribution drastically reduces the noise in the data, especially towards the end of the sequence as it can leverage information from many data points. The smoothed posterior distribution manages to accurately estimate the trajectory and reduce the noise at all time steps, as it can use data from the whole sequence. As expected, its trajectory it is much smoother that the one estimated by the Kalman filter. Finally, notice that the filter

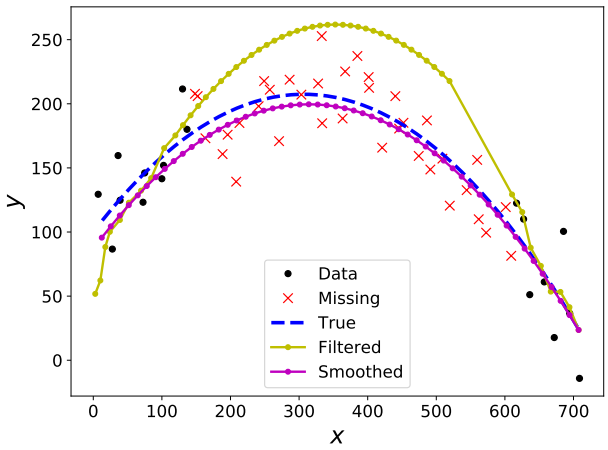



***Figure 3.4:*** *Missing data imputation in the ball tracking example.*

manages to correct the choice of initial state, that was chosen on purpose to be far from the true one. This is possible as the emission noise was chosen to be relatively small, in which case at the first time step the Kalman gain will be big and give more importance to the true observation.

### 3.4.2 Missing data imputation

In many applications we have time series that have been observed irregularly over time. An appealing properties of LGSSM is that they allow to deal with missing data in a simple and principled way. We can in fact use the original Kalman filtering and smoothing algorithms at all time steps taking $\mathbf{K}_t = \mathbf{0}$ (or equivalently $\mathbf{C}_t = \mathbf{0}$) at the missing time points. In this case, the measurement step will only propagate the sufficient statistics computed during the prediction step, i.e. we will have

$$\begin{aligned} \boldsymbol{\mu}_t &= \boldsymbol{\mu}_{t|t-1} \\ \boldsymbol{\Sigma}_t &= \boldsymbol{\Sigma}_{t|t-1} \ , \end{aligned}$$

so that the filtered posterior coincides with the predictive prior. Once the filtered or smoothed posterior are computed, the missing observation can be estimated simply as $\mathbf{C}_t \widehat{\mathbf{z}}_t$, with $\widehat{\mathbf{z}}_t$ being the inferred value. In a similar way, the algorithms can be extended to *partially observed* $\mathbf{x}_t$, i.e. observations $\mathbf{x}_t$ with missing elements, by not considering in the emission matrix the rows corresponding to a missing element, see (Durbin and Koopman, 2012) for details.

We finally notice that predictions with a LGSSM can be obtained by treating future observations as missing values, simply running the Kalman filtering algorithm setting $\mathbf{K}_t = \mathbf{0}$ for the future time steps that we want to predict.

**Ball tracking example.** We consider the same trajectory as in Figure 3.3, but we now only observe the first 10 and last 8 observations, treating all the central ones as missing. In Figure 3.4 we plot the estimated trajectory from the Kalman filter and smoother. We see that the filtered posterior does not provide a good estimate of the trajectory when the data is missing, as the

first 10 time steps only do not provide enough reliable information (the estimate recovers only with the last 8 observations around $x = 600$). As opposed to the filtered posterior, the smoothed one can use the information on the last 8 observations at all time steps. The smoothed posterior is therefore very accurate, and close to the one obtained in Figure 3.3 when all the data was observed.

### 3.4.3 Parameter learning with the EM algorithm

In some cases, some or all of the parameters of the LGSSM are unknown, and we need to learn them from data. Apart from having parameters to learn we also have latent variables to infer, which is the scenario for which in Section 2.3.1 we introduced the EM algorithm.[3] The EM algorithm for LGSSMs was first introduced in (Shumway and Stoffer, 1982) assuming that the emission matrix was known, and then generalized in (Ghahramani and Hinton, 1996) to models where all parameters were unknown. The E-step for a LGSSM can be easily done exploiting the tractability of the filtered and smoothed posteriors, and the optimization of all variables in the M-step has a simple closed form solution. In this section we will see as an example how we can learn the parameters of the prior $p(\mathbf{z}_1)$ of the initial state. Detailed derivations for all the parameters of the emission and transition distributions of the LGSSM can be found in (Ghahramani and Hinton, 1996) or (Bishop, 2006). A discussion on the identifiability of the model when learning all the parameters can be found in (Roweis and Ghahramani, 1999).

The parameters $\boldsymbol{\mu}_{1|0}$ and $\boldsymbol{\Sigma}_{1|0}$ of the initial state Gaussian prior can be optimized by maximizing the ELBO in (2.7). In the E-step of (2.8) we maximize the ELBO with respect to the variational approximation while keeping the parameters fixed. The optimal distribution for the LGSSM is given by the smoothed posterior marginal at the first time step, i.e. $q_{k+1}(\mathbf{z}_1) = p_\theta(\mathbf{z}_1|\mathbf{x}_{1:T}, \mathbf{u}_{1:T})$. In the M-step of (2.9) we then keep the posterior distribution fixed, and maximize the ELBO with respect to the parameters of the initial state prior. In this maximization, all the terms that do not depend on $\boldsymbol{\mu}_{1|0}$ or $\boldsymbol{\Sigma}_{1|0}$ can be absorbed in an additive constant, giving the ELBO:

$$\mathcal{F}(\boldsymbol{\mu}_{1|0}, \boldsymbol{\Sigma}_{1|0}) = -\frac{1}{2}\log\det(\boldsymbol{\Sigma}_{1|0}) - \mathbb{E}_{p_\theta(\mathbf{z}_1|\mathbf{x}_{1:T},\mathbf{u}_{1:T})}\left[\frac{1}{2}(\mathbf{z}_1 - \boldsymbol{\mu}_{1|0})^T\boldsymbol{\Sigma}_{1|0}^{-1}(\mathbf{z}_1 - \boldsymbol{\mu}_{1|0})\right] + \text{ constant}$$

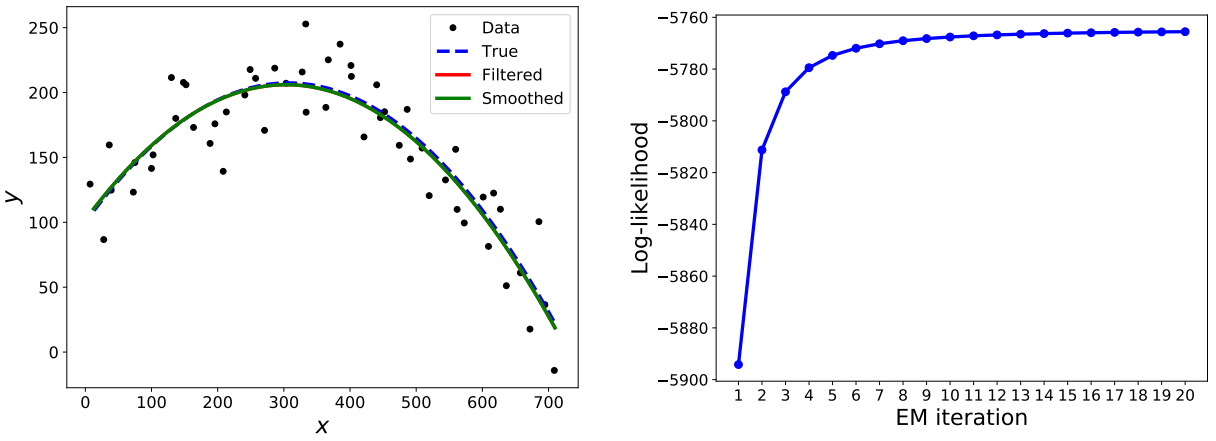
where $\det(\cdot)$ denotes the matrix determinant and the superscript $T$ the transpose operation. By setting to 0 the partial derivatives of $\mathcal{F}(\boldsymbol{\mu}_{1|0}, \boldsymbol{\Sigma}_{1|0})$ with respect to each of the parameters we obtain the M-step updates

$$\begin{aligned}\boldsymbol{\mu}_{1|0}^{new} &= \boldsymbol{\mu}_{1|T} \\ \boldsymbol{\Sigma}_{1|0}^{new} &= \boldsymbol{\Sigma}_{1|T} - \boldsymbol{\mu}_{1|T}\boldsymbol{\mu}_{1|T}^T \,.\end{aligned}$$

This new initial state can now be used to compute the smoothed posterior needed for the E-step in the following iteration of the EM algorithm, that is let to run until convergence.

As we have seen in (3.12), as a byproduct of the Kalman filtering algorithm we can compute the marginal likelihood of the model. An alternative approach to ML learning of the parameters of the LGSSM is then the direct numerical optimization of this quantity with gradient-based methods such as the L-BFGS algorithm (Liu and Nocedal, 1989). Shumway and Stoffer, (1982) suggest to

[3] The same equations introduced in Section 2.3.1 apply, if we consider $\mathbf{x} = \mathbf{x}_{1:T}$ and $\mathbf{z} = \mathbf{z}_{1:T}$.




*(a) Inferred states with learned initial state.*

*(b) Log-likelihood.*

***Figure 3.5:*** *Parameter learning in the ball tracking example.*

use the EM algorithm for the first iterations of the learning procedure and switch to gradient-based methods after some iterations, arguing that the EM algorithm has a slow convergence when the parameters approach the local optimum but is more robust than gradient-based methods to poor parameter initializations.

In this discussion we have assumed that the parameters of the LGSSM are unknown but fixed quantities. It is however possible to consider the parameters as random variables, that results in the *Bayesian LGSSM*. This Bayesian approach is particularly useful when we need to include strong prior constraints in the parameters to be able to find suitable solutions. One problem that arises in this case is that state inference becomes intractable, and one needs to resort to approximate inference techniques such as the variational EM algorithms of Beal, (2003); Barber and Chiappa, (2007).

**Ball tracking example.** In our description of the ball tracking example in Section 3.4 we mentioned that the choice of the prior over the initial state was suboptimal. We now use the EM algorithm to learn the parameters of the initial state $\boldsymbol{\mu}_{1|0}$ and $\boldsymbol{\Sigma}_{1|0}$, running it for 20 iterations and initializing the parameters as before. In Figure 3.5a, we plot the state inferred with the Kalman filter and smoother after the optimization. We see that now both the filtered and smoothed trajectories match very closely the ground truth one. In Figure 3.5b we plot the change in log-likelihood of the model during the EM algorithm. As discussed in Section 2.3.1, as inference during the E-step is exact, the log-likelihood does not decrease after each EM iteration.

## 3.5 Non-linear non-Gaussian state-space models

While the LGSSM provides an elegant mathematical framework for exact state inference, missing data imputation and parameter learning, its applicability is limited by the fact that in many cases the linear-Gaussian assumptions for the transitions and emissions are too strong. These assumptions are no longer valid for example if we consider small extensions to the ball tracking

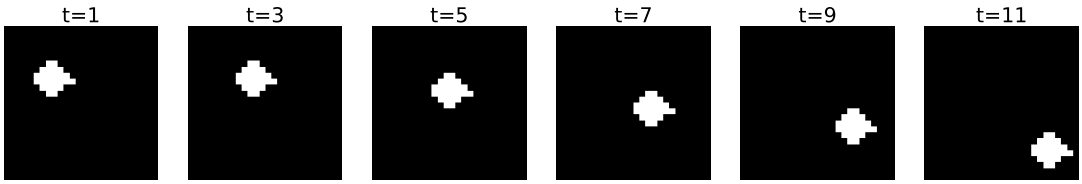



***Figure 3.6:*** *Tracking a ball in a video. At each time step $t$ we observe a single frame of the video.*

example discussed in Section 3.4:

- *Non-linear transitions.* To make the transition linear we had to assume that the ball was traveling in vacuum. In a more realistic setting however the ball moves in air, therefore we also need to take into account air resistance, which adds a non-linear relationship between the velocities at consecutive steps and would strongly modify the final trajectory. Other non-linearities could be given by obstacles in the trajectory of the ball, e.g. walls, see (Fraccaro et al., 2017, Chapter 6) for an example.

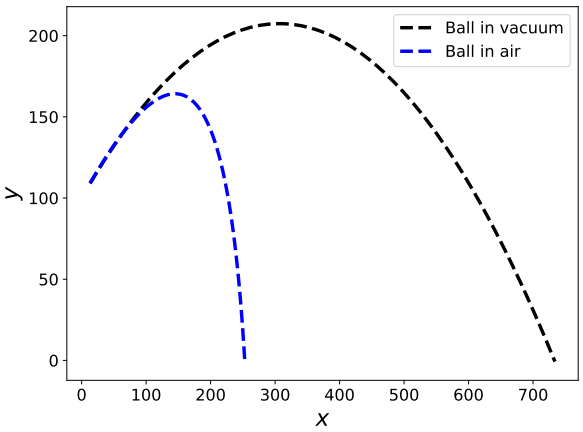
- *Non-Gaussian transition noise.* In a real world scenario a precise model of the ball would take into account many physical effects such as the spin of the ball, its roughness, the altitude and the humidity of the air. As it would be very difficult to find a precise mathematical model for these we may want to treat them as noise. These effects would slightly but consistently modify the trajectory of the ball, therefore a Gaussian noise would probably be a poor choice for this (we may need for example a skewed distribution).

- *Non-linear emissions.* To have linear emissions, we assumed that we had a noisy sensor that could track the 2-dimensional $(x, y)$ position of the ball. In (Fraccaro et al., 2017, Chapter 6) we will assume that our sensor is a camera, therefore instead of observing the positions over time we observe a video of the ball flying in the vacuum. An example of a sequence of observations is given in Figure 3.6. In this case our observations will be high-dimensional images, that can only be modelled accurately with non-linear emissions, e.g. parameterized with convolutional neural networks.

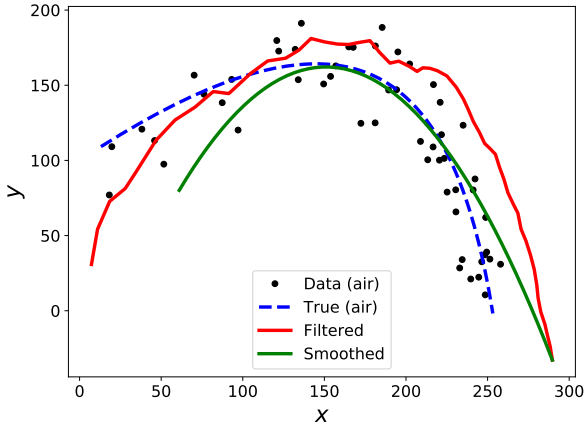
- *Non-Gaussian emission noise.* In the ball tracking example we have assumed that the emission noise is Gaussian. Inaccurate sensor however may return many outliers, that would be better modelled with a heavy-tailed distribution such as a Student-t distribution.

In all these cases suitable emission distribution $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$ and transition distribution $p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)$ will be non-linear and/or non-Gaussian. The main issue that arises in this case is that inference and parameter learning become intractable, and will need to be approximated as shown in Section 3.5.1.

**Ball tracking example with air resistance.** We now illustrate how the trajectory of the ball changes if we assume that it travels in air instead of vacuum (Labbe, 2015). To model the physics of this system, the velocities in the equations of motions in (3.9) need to be modified by




*(a) Comparison between trajectory in vacuum and in air.*




*(b) Posterior inference with a LGSSM.*

***Figure 3.7:*** *Ball tracking example with air resistance.*

subtracting two extra non-linear components that take into account the air resistance:

$$\begin{cases} x_t &= x_{t-1} + \dot{x}_{t-1}\Delta \\ \dot{x}_t &= \dot{x}_{t-1} - 0.0039 + \frac{0.0058}{1+e^{\left(\sqrt{\dot{x}_{t-1}^2+\dot{y}_{t-1}^2}-35\right)/5}}\dot{x}_{t-1}\Delta \\ y_t &= y_{t-1} + \dot{y}_{t-1}\Delta - \frac{1}{2}g\Delta^2 \\ \dot{y}_t &= \dot{y}_{t-1} - g\Delta - 0.0039 + \frac{0.0058}{1+e^{\left(\sqrt{\dot{x}_{t-1}^2+\dot{y}_{t-1}^2}-35\right)/5}}\dot{y}_{t-1}\Delta \ . \end{cases} \tag{3.13}$$

A detailed derivation of these equations is out of the scope of this thesis, for which we are only interested in their non-linear nature.

In Figure 3.7a we see that even if we use the same initial conditions the trajectory is no longer a parabola, since the air resistance slows down the ball in a non-linear way. In Figure 3.7b we apply the same LGSSM introduced in Section 3.4 to this new data. As expected, the linear transitions of the LGSSM cannot properly model the non-linear effect of the air resistance, resulting in a poor state estimation. We will see in the next section that this issue can be corrected using non-linear models.

### 3.5.1 Approximate inference and learning

We now provide a brief overview of the main algorithms developed over the years for approximate inference and learning in non-linear non-Gaussian SSMs. A more in depth treatment of both deterministic and stochastic approximation techniques can be found in (Barber et al., 2011) and (Särkkä, 2013). In Section 4.3.1 we will then discuss variational methods for non-linear SSMs parameterized by deep neural networks.

#### 3.5.1.1 Extended Kalman Filter

We now consider non-linear models with Gaussian noise, in which the transition and emission equations are described by two non-linear *differentiable* functions, denoted as $f$ and $g$ respectively:

$$\begin{aligned}\mathbf{z}_t &= f(\mathbf{z}_{t-1}, \mathbf{u}_t) + \boldsymbol{\varepsilon}_t \\ \mathbf{x}_t &= g(\mathbf{z}_t) + \boldsymbol{\delta}_t\end{aligned}$$

with $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(\boldsymbol{\varepsilon}_t; \mathbf{0}, \mathbf{Q}_t)$ and $\boldsymbol{\delta}_t \sim \mathcal{N}(\boldsymbol{\delta}_t; \mathbf{0}, \mathbf{R}_t)$. The *Extended Kalman Filter* (EKF) (Smith et al., 1962) is a deterministic approximation technique that can be used to compute a Gaussian approximation to the posterior distribution for this class of models. It works by linearizing both functions around the estimated posterior mean using a Taylor series expansion, and applying the standard Kalman filtering and smoothing algorithms in this new linearized space.

We linearize the transition equation around the previous state estimate $\boldsymbol{\mu}_{t-1}$:

$$f(\mathbf{z}_{t-1}, \mathbf{u}_t) = f(\boldsymbol{\mu}_{t-1} + (\mathbf{z}_{t-1} - \boldsymbol{\mu}_{t-1}), \mathbf{u}_t) \approx f(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) + \mathbf{F}_t(\mathbf{z}_{t-1} - \boldsymbol{\mu}_{t-1})$$

where $\mathbf{F}_t$ is the Jacobian matrix of $f$ evaluated at $\boldsymbol{\mu}_{t-1}$.

$$\mathbf{F}_t = \frac{\partial}{\partial \mathbf{z}_{t-1}} f(\mathbf{z}_{t-1}, \mathbf{u}_t)\Big|_{\mathbf{z}_{t-1}=\boldsymbol{\mu}_{t-1}, \mathbf{u}_t}$$

In a similar way, we linearize the emission equation around the predictive prior mean $\boldsymbol{\mu}_{t|t-1}$:

$$\begin{aligned}g(\mathbf{z}_t) = g(\boldsymbol{\mu}_{t|t-1} + (\mathbf{z}_t - \boldsymbol{\mu}_{t|t-1})) &\approx g(\boldsymbol{\mu}_{t|t-1}) + \mathbf{G}_t(\mathbf{z}_t - \boldsymbol{\mu}_{t|t-1}) \\ \mathbf{G}_t = \frac{\partial}{\partial \mathbf{z}_t} g(\mathbf{z}_t)\Big|_{\mathbf{z}_t=\boldsymbol{\mu}_{t|t-1}} &.\end{aligned}$$

The resulting transition and emission distributions are now linear with respect to $\mathbf{z}_{t-1}$ and $\mathbf{z}_t$, i.e.

$$\begin{aligned}p_{\theta_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) &= \mathcal{N}(\mathbf{z}_t; f(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) + \mathbf{F}_t(\mathbf{z}_{t-1} - \boldsymbol{\mu}_{t-1}), \mathbf{Q}_t) \\ p_{\theta_t}(\mathbf{x}_t|\mathbf{z}_t) &= \mathcal{N}(\mathbf{x}_t; g(\boldsymbol{\mu}_{t|t-1}) + \mathbf{G}_t(\mathbf{z}_t - \boldsymbol{\mu}_{t|t-1}), \mathbf{R}_t)\ ,\end{aligned}$$

and we can then apply the standard Kalman filter and smoothing algorithms. The computational complexity of the EKF is therefore similar to the one of the Kalman filter, apart from the computations needed for the non-linear functions $f$ and $g$ and their Jacobians. Notice that the EKF algorithm reduces to the standard Kalman filter if we use $f(\mathbf{z}_{t-1}, \mathbf{u}_t) = \mathbf{A}_t\mathbf{z}_{t-1} + \mathbf{B}_t\mathbf{u}_t$ and $g(\mathbf{z}_t) = \mathbf{C}_t\mathbf{z}_t$, i.e. we have a LGSSM.

The EKF can also be used as E-step when learning the parameters of the model, as done in (Ghahramani and Roweis, 1999) for models in which Gaussian radial basis functions are used to model non-linearities.

#### 3.5.1.2 The Unscented Kalman Filter

Due to the linearization steps, the EKF is mostly suitable for systems that are almost linear, as it may quickly diverge with highly non-linear $f$ and $g$. The *Unscented Kalman Filter* (UKF) (Julier

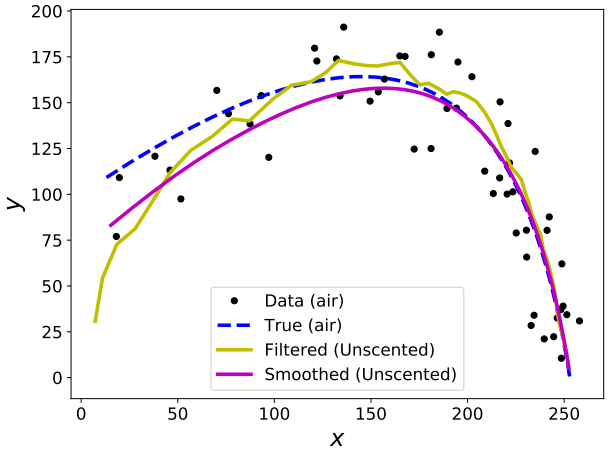



***Figure 3.8:*** *Tracking the ball with a non-linear model and the Unscented Kalman filter/smoother.*

and Uhlmann, 1997) works similarly to the EKF, but provides a different way to approximate Gaussian random variables that performs better with highly non-linear transitions and emissions. Here we will give the main intuition behind the algorithm and an example of application, see (Murphy, 2012) for a thorough derivation. Instead of first approximating the non-linear functions by linearization and then passing a Gaussian distribution through it as in the EKF, the UKF first passes a deterministically chosen set of points through the non-linear functions and then approximates the resulting distribution with a Gaussian. This set of points, called *sigma points*, are deterministically sampled and chosen to capture the mean and covariance of the Gaussian random variables. The UKF is typically more accurate and robust than the EKF, and has a similar computational complexity. Furthermore, the UKF does not require derivatives, and can therefore work even for non-differentiable functions. As shown in (Wan and van der Merwe, 2001; Särkkä, 2008), the UKF can also be extended to perform smoothing.

**Ball tracking example with air resistance.** We start from the same LGSSM used in Section 3.4, but we make the transition transition equation $f$ of the SSM non-linear as in (3.13). In Figure 3.8 we then use the Unscented Kalman filter and smoother to perform state inference. As expected, we see that thanks to the non-linear formulation of the model the estimated trajectory is much closer to the true one compared to the one obtained the LGSSM of Figure 3.7b.

### 3.5.1.3 Importance sampling

*Importance sampling* is a sampling technique that can be used to approximate posterior expectations as well as to find a sample-based approximation to the posterior $p_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$ (Doucet et al., 2001).

Consider the task of finding the posterior expectation of a generic function of the latent states $h(\mathbf{z}_{1:t})$, i.e. computing

$$\mathbb{E}(h) = \mathbb{E}_{p_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t},\mathbf{u}_{1:t})}\left[h(\mathbf{z}_{1:t})\right] = \int h(\mathbf{z}_{1:t}) p_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) \mathrm{d}\mathbf{z}_{1:t} \ . \tag{3.14}$$

For example, solving this integral with $h(\mathbf{z}_{1:t}) = \mathbf{z}_{1:t}$ would return the posterior mean. The posterior distribution over which we want to take the expectation is intractable for non-linear non-Gaussian models, for which we can only evaluate the unnormalized posterior (i.e. the joint distribution $p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|\mathbf{u}_{1:t})$). We can however find an approximation to this integral by introducing an auxiliary distribution $q(\mathbf{z}_{1:t})$, called *importance distribution*, as we did with the variational distribution that we used to define the ELBO in (2.7). To stress the fact that the importance distribution may depend on $\mathbf{x}_{1:t}$ and $\mathbf{u}_{1:t}$, we denote it as $q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$.

For any distribution $q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$ whose support includes the support of $p_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$, we can rewrite the integral in (3.14) as

$$\begin{aligned}
\mathbb{E}(h) &= \int h(\mathbf{z}_{1:t}) \frac{p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|\mathbf{u}_{1:t})}{p_\theta(\mathbf{x}_{1:t}|\mathbf{u}_{1:t})} \mathrm{d}\mathbf{z}_{1:t} \\
&= \frac{\int h(\mathbf{z}_{1:t}) \frac{p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|\mathbf{u}_{1:t})}{q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})} q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) \mathrm{d}\mathbf{z}_{1:t}}{\int \frac{p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|\mathbf{u}_{1:t})}{q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})} q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) \mathrm{d}\mathbf{z}_{1:t}} \\
&= \frac{\int h(\mathbf{z}_{1:t}) w(\mathbf{z}_{1:t}) q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) \mathrm{d}\mathbf{z}_{1:t}}{\int w(\mathbf{z}_{1:t}) q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) \mathrm{d}\mathbf{z}_{1:t}}
\end{aligned} \tag{3.15}$$

where we have defined the *unnormalized importance weights* $w(\mathbf{z}_{1:t})$ as

$$w(\mathbf{z}_{1:t}) = \frac{p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|\mathbf{u}_{1:t})}{q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})} \ . \tag{3.16}$$

If we choose an importance distribution which is easy to sample from, e.g. a multivariate Gaussian, we can find a sample-based approximation to the integral in (3.14). We first draw $R$ i.i.d. samples $\mathbf{z}_{1:t}^{(r)} \sim q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$, that are also referred to as *particles* in this setting, and successively approximate the expectation with Monte Carlo integration:

$$\widehat{\mathbb{E}}(h) = \frac{\frac{1}{R}\sum_{i=1}^{R} w(\mathbf{z}_{1:t}^{(r)}) h(\mathbf{z}_{1:t}^{(r)})}{\frac{1}{R}\sum_{i=1}^{R} w(\mathbf{z}_{1:t}^{(r)})} = \sum_{i=1}^{R} \widetilde{w}_t^{(r)} h(\mathbf{z}_{1:t}^{(r)}) \ , \tag{3.17}$$

where we have defined the *normalized importance weights* as

$$\widetilde{w}_t^{(r)} = \frac{w(\mathbf{z}_{1:t}^{(r)})}{\sum_{i=1}^{R} w(\mathbf{z}_{1:t}^{(r)})} \ .$$

The estimate $\widehat{\mathbb{E}}(h)$ is therefore computed with a weighted average, whose weights depend on the ratio between the densities of $p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|\mathbf{u}_{1:t})$ and $q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$. If $q$ is smaller than $p$ at a certain state $\mathbf{z}_{1:t}$ the corresponding weight will be high, vice versa the weight will be small if $q$ is larger than $p$. This allows to correct in the estimate $\widehat{\mathbb{E}}(h)$ the mismatch between the two distributions, making sure that we are correctly representing the high-probability regions of $p$ despite the fact we are sampling from $q$. Asymptotically, as $R \to \infty$ the estimate $\widehat{\mathbb{E}}(h)$ will converge to $\mathbb{E}(h)$. In practice however we can only use a finite number of samples, and this makes this estimate biased and only accurate if the choice of the specific form of the importance distribution is sufficiently close to the posterior distribution (so that most of the particles fall in the high probability regions of $p$).

Notice that the set of weight-particle pairs $\{(\widetilde{w}_t^{(r)}, \mathbf{z}_{1:t}^{(r)})\}_{i=1}^R$ can be seen as a sample-based approximation of the posterior $p_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$, defined as a weighted mixture of delta functions centered at the samples:

$$\widehat{p}_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) = \sum_{i=1}^{R} \widetilde{w}_t^{(r)} \delta(\mathbf{z}_{1:t} - \mathbf{z}_{1:t}^{(r)}) \ .$$

Using this approximation we can in fact rewrite (3.17) as

$$\widehat{\mathbb{E}}(h) = \int h(\mathbf{z}_{1:t}) \widehat{p}_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) \mathrm{d}\mathbf{z}_{1:t} \ .$$

One issue with importance sampling is that it is highly inefficient in high dimensional cases. Also, in the sequential setting this algorithm is not suitable for recursive estimation of the filtered posterior distribution. At each new time step we would have in fact to compute the importance weights over the whole sequence of states, i.e. the computational complexity increases over time. We now present a sequential extension of this algorithm that mitigates both of these problems.

#### 3.5.1.4 Particle filtering

In Section 3.4.1 we have seen that the Kalman filter provides a closed-form expression for the recursive estimation of the marginal posterior distribution $p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$ given the one at the previous time step, $p_\theta(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1})$. *Particle filtering* (Doucet et al., 2001; Doucet and Johansen, 2008), on the other hand, uses a sequential extension of importance sampling to recursively update a numerical approximation to the posterior over the whole sequence up to time $t$, i.e. $p_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$, given the posterior $p_\theta(\mathbf{z}_{1:t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1})$ at time steps $1:t-1$. Using the laws of probability as well as the conditional independence properties of SSMs, we can in fact decompose the posterior $p_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$ as

$$\begin{aligned}
p_\theta(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) &= \frac{p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|, \mathbf{u}_{1:t})}{p_\theta(\mathbf{x}_{1:t}|\mathbf{u}_{1:t})} \\
&= \frac{p_\theta(\mathbf{x}_t, \mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}, \mathbf{u}_{1:t}) p_\theta(\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}|, \mathbf{u}_{1:t-1})}{p_\theta(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) p_\theta(\mathbf{x}_{1:t-1}|\mathbf{u}_{1:t-1})} \\
&= \frac{p_\theta(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{u}_{1:t}) p_\theta(\mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}, \mathbf{u}_{1:t})}{p_\theta(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t})} p_\theta(\mathbf{z}_{1:t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1}) \\
&= \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t) p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)}{p(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t})} p_\theta(\mathbf{z}_{1:t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1}) \ .
\end{aligned}$$

To allow for recursive estimation, we further assume the following factorization for the importance distribution

$$q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) = q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t}, \mathbf{u}_{1:t}) q(\mathbf{z}_{1:t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1}) = q(\mathbf{z}_1) \prod_{l=2}^{t} q(\mathbf{z}_l|\mathbf{z}_{1:l-1}, \mathbf{x}_{1:l}, \mathbf{u}_{1:l}) \ .$$

The unnormalized importance weights then become

$$\begin{aligned} w(\mathbf{z}_{1:t}) &= \frac{p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|, \mathbf{u}_{1:t})}{q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})} \\ &= \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t)p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)p_\theta(\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}|, \mathbf{u}_{1:t-1})}{q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t}, \mathbf{u}_{1:t})q(\mathbf{z}_{1:t-1}|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1})} \\ &= \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t)p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)}{q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t}, \mathbf{u}_{1:t})} w(\mathbf{z}_{1:t-1}) \end{aligned}$$

which implies that the normalized importance weights are

$$\widetilde{w}_t^{(r)} \propto \underbrace{\frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t)p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)}{q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t}, \mathbf{u}_{1:t})}}_{\alpha_t} \widetilde{w}_{t-1}^{(r)} \ . \tag{3.18}$$

The importance weight at time $t$ can then be simply computed given the ones at the previous time step by multiplying the *incremental importance weights* $\alpha_t$. The computational complexity of the algorithm stays therefore constant over time.

Starting from a set of particles $\{(\widetilde{w}_{t-1}^{(r)}, \mathbf{z}_{1:t-1}^{(r)})\}_{i=1}^R$, that provides an empirical estimate of the posterior at times $1 : t-1$, we can obtain an approximation of the posterior at times $1 : t$ by extending each particle sampling from $q(\mathbf{z}_t|\mathbf{z}_{1:t-1}^{(r)}, \mathbf{x}_{1:t}, \mathbf{u}_{1:t})$ and updating the corresponding weight using (3.18). This procedure can be initialized by sampling $R$ i.i.d particles from $q(\mathbf{z}_1)$.

The algorithm discussed until now is also known as *sequential importance sampling*. A particle filter combines sequential importance sampling with a *resampling* step that is added to prevent *degeneracy* in the particles. In practice in fact, as $t$ increases the distribution of the weights will become skewed, with just a small percentage of the particles having a non-zero weight. Only a few particles will then effectively approximate the posterior distribution, and a lot of computational resources will be wasted on particles that have a negligible effect on the approximation of the expectation (3.17). However, if at each iteration we resample with replacement the $R$ particles using their weight as resampling probabilities, particles with higher weight will be replicated while particles with low weight will be discarded. This will result in a new set of $R$ particles, that will be assigned weight $\frac{1}{R}$. A discussion on different resampling methods can be found in (Doucet and Johansen, 2008). While beneficial to prevent degeneracy in the particles, this resampling step is introduces some variance in the estimate and should therefore be done only if necessary. Liu and Chen, (1998) provide the *effective sample size*

$$R_{eff} = \frac{1}{\sum_{r=1}^R (\widetilde{w}_t^{(r)})^2}$$

as a metric to measure of the variance of the weights and suggest whether the resampling step should be done. In practice the resampling step is only done if the effective sample size is below a certain number of particles, e.g. $R_{eff} < 0.7R$.

The choice of the importance distribution $q$ is crucial to the success of the particle filter. The optimal distribution in order to minimize the variance of the importance weights is given by

$$q_{opt}(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t}, \mathbf{u}_{1:t}) = p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_t, \mathbf{u}_t) = \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t)p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)}{p_\theta(\mathbf{x}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)} \tag{3.19}$$

In practice this distribution is often intractable, and many approximations have been proposed over the years. Among the simplest ones, the *bootstrap filter* uses the prior distribution as importance distribution

$$q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t}) = p_\theta(\mathbf{z}_{1:t}|\mathbf{u}_{1:t}) = p(\mathbf{z}_1) \prod_{l=2}^{t} p_\theta(\mathbf{z}_l|\mathbf{z}_{l-1}, \mathbf{u}_l)$$

so that $\widetilde{w}_t^{(r)} \propto p_\theta(\mathbf{x}_t|\mathbf{z}_t)\widetilde{w}_{t-1}^{(r)}$. In (Doucet et al., 2000; van der Merwe et al., 2000) the authors propose to use the approximate Gaussian posterior obtained with the EKF and UKF as proposal distributions. As we will see in Section 4.3.2, it is also possible to learn flexible importance distributions parameterized by deep neural networks.

We finally notice that as a byproduct of this inference procedure we can also obtain an unbiased estimate of the marginal likelihood $p_\theta(\mathbf{x}_{1:t}|\mathbf{u}_{1:t})$, the denominator in (3.15), using the intermediate unnormalized weights:

$$\widehat{p}_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T}) = \prod_{t=1}^{T} \frac{1}{R} \sum_{r=1}^{R} w(\mathbf{z}_{1:t}^{(r)}) \ . \tag{3.20}$$

Particle fiiltering is in general more computationally demanding but more accurate than the EKF and UKF. It is a special case of the broader class of Sequential Monte Carlo algorithm (Del Moral et al., 2006), and borrows from it many of the techniques developed to improve sampling. For example, MCMC steps can be used after a resampling step to avoid having too may identical copies of the same particle, i.e. sample impoverishment (Gilks and Berzuini, 2001). Also, the number of particles can be adaptively chosen depending on the effective sample size (Fraccaro et al., 2016a).

These ideas can be generalized to compute the smoothed posterior distribution (*particle smoothing*) (Doucet and Johansen, 2008) as well as to estimate the parameters of the model, e.g. with the EM algorithm or gradient-based methods (Kantas et al., 2015).

## 3.6 Summary and discussion

In this chapter we have introduced state-space models as an extension of LVMs that is suitable for sequential data. The Markovian structure of SSMs introduces some conditional independence properties between the latent variables that can be exploited during inference (filtering, smoothing and prediction). We have presented the LGSSM as a basic example of a SSM, for which posterior inference and missing data imputation can be performed in an exact way. Non-linear and non-Gaussian SSMs can be used to model more complex sequences, but require approximate inference procedures such as the EKF, the UKF or particle filters. We have used the ball tracking example to illustrate many of the techniques presented throughout this chapter.

The main focus of this thesis is building non-linear models that can learn complex high-dimensional sequential data distributions from large unlabelled datasets. The approximate inference methods presented in Section 3.5.1 are however not powerful and/or scalable enough for such applications. In the next chapter we will therefore introduce a general class of models that use SSMs with

highly non-linear transitions and emissions parameterized by deep neural networks. These models are broadly applicable and can be trained efficiently with the amortized Variational inference ideas presented for VAEs in Section 2.4.

Chapter 4

# Deep latent variable models for sequential data

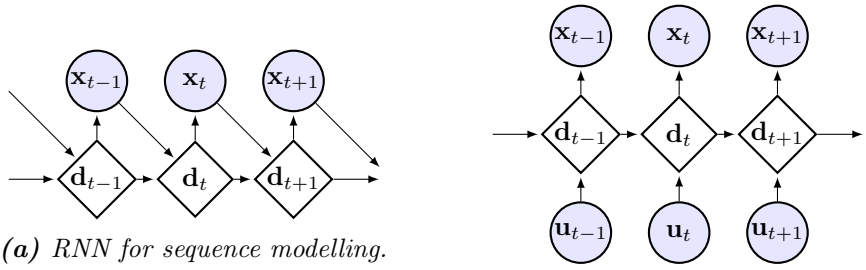
## 4.1 Motivation

The main focus of this thesis is unsupervised learning of complex probability distributions for temporal data. We may be interested for example in learning a generative models for speech, music, videos or text, or in using the data stored in electronic health records (EHRs) to learn a patient representation given the information collected during many different visits. These applications are characterized by:

1. *Complex and high-dimensional temporal distributions.* We consider high-dimensional sequences (e.g. a video), that require complex architectures that are able to:
   - Model the high-dimensional observations at each time step.
   - Capture long-term temporal dependencies in the data and memorize relevant information.
   - Model the uncertainty and variability in the data, and properly propagate them over time.

   These models will have an intractable log-likelihood, and we will need to resort to approximate inference and parameter learning.
2. *Large-scale datasets.* To learn such complex distributions that possibly depend on hundreds of thousands of parameters we will use very large datasets. We then need scalable models and training procedures.

We will solve these tasks by combining ideas from three classes of models closely related to each other. First, as we have see in Chapter 2, we can use **VAEs** to model complex high dimensional




***(a)*** *RNN for sequence modelling.*

***(b)*** *RNN with input variables.*

***Figure 4.1:*** *A graphical representation of a recurrent neural network. Diamond-shaped units denote deterministic states.*

observations by introducing latent variables and using neural networks to parameterize flexible conditional distributions. VAEs allow to perform training using stochastic back-propagation with inference networks in a very scalable way. Secondly, we can use recurrent neural networks (**RNNs**), that will be introduced in Section 4.2, to model long-term dependencies in the data through their parametric memory cells. Finally, we will show in Section 4.3 that the same ideas that lead to develop VAEs as a deep LVM can be applied to construct **deep SSMs**, flexible and scalable models for temporal data that offer a principled way to model uncertainty in the latent representation.

As we will see in the rest of the Chapter, depending on the needs of each application these models can be combined in many different ways. In Section 4.4 we will see that we can extend VAEs to sequential data by using an RNN to define a time-varying prior for the same VAE repeated at each time step. These models can then be further extended using a DSSM instead of the VAEs, as shown in Section 4.5. In Section 4.6 we will then show how these ideas can be used to learn disentangled representations by defining structured prior distribution. In some applications that require higher memory capacity RNNs are not enough, and we need therefore to extend the sequential models using external memory architectures as discussed in Section 4.7.

All these models use neural networks as their main building block, and we will therefore be able to define very expressive and flexible architectures that can be trained in a similar way using stochastic back-propagation and are simple to implement using existing deep learning libraries. Due to their expressiveness it is not always easy to fully exploit the modelling power of these architectures. In Section 4.8 we will then present several training tricks that have proven useful in many applications.

## 4.2 Recurrent neural networks

*Recurrent neural networks (RNN)* are an extension of deep neural networks that can model sequences of variable length. They are widely used in many areas involving temporal data, such as for language modelling and machine translation (Graves, 2013; Sutskever et al., 2014).

For sequence modelling, RNNs assume the following factorization of the joint distribution over

the sequence $\mathbf{x}_{1:T}$,

$$p_\theta(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{x}_{1:t-1}) \ , \tag{4.1}$$

and assume that at time $t$ all the relevant information coming from the past that is included in $\mathbf{x}_{1:t-1}$ can be summarized by a deterministic latent variable $\mathbf{d}_t$. A graphical representation of the model can be found in Figure 4.1a. Instead of defining $p_\theta(\mathbf{x}_t|\mathbf{x}_{1:t-1})$, in an RNN we define $p_\theta(\mathbf{x}_t|\mathbf{d}_t)$, as a distribution whose parameters depend on $\mathbf{d}_t$ through some deep neural networks parametrized by $\theta$ (e.g. a Gaussian distribution). The *state* $\mathbf{d}_t$ evolves over time, and at each time step incorporates the information from the previous element of the sequence, using the state update equation $\mathbf{d}_t = f_\theta(\mathbf{d}_{t-1}, \mathbf{x}_{t-1})$. The function $f_\theta$ is a differentiable non-linear transition function that has to be powerful enough to capture the long-term dependencies in the data. Common choices for $f_\theta$ are memory cell units such as LSTMs (Hochreiter and Schmidhuber, 1997) or GRU (Chung et al., 2014), that use learned gating mechanisms to store information that needs to be available at future time steps. The initial state $\mathbf{d}_0$ of the RNN is typically learned or set to $\mathbf{0}$. The RNN is trained to predict the next output of the sequence given all the previous ones, and gradients can be computed using a temporal extension to the back-propagation algorithm.

Notice that more in general RNNs can be used to model sequences $\mathbf{x}_{1:T}$ that depend on some inputs $\mathbf{u}_{1:T} = [\mathbf{u}_1, \dots, \mathbf{u}_T]$, as illustrated in Figure 4.1b. In this case, the state update equation is given by $\mathbf{d}_t = f_\theta(\mathbf{d}_{t-1}, \mathbf{u}_t)$, i.e. $\mathbf{d}_t$ is now used to capture at each time step the information coming from the input $\mathbf{u}_t$. When doing sequence modelling, as illustrated in Figure 4.1a we are implicitly considering the output at the previous time step as the input ($\mathbf{u}_t = \mathbf{x}_{t-1}$), which is allowed as at time $t$ the value of $\mathbf{x}_{t-1}$ is known (if we assume that there are no missing values). In the following, even when considering sequence modelling we will use $\mathbf{u}_t$ in the equations, as it results in a more general as well as cleaner notation.

By comparing the graphical representation of SSMs and RNNs in Figures 3.1 and 4.1b respectively, it is easy to see that an RNN can be interpreted as a special case of a SSM whose transition distribution is a delta function that expresses a highly non-linear but deterministic relationship between the RNN states. As the RNN units $\mathbf{d}_t$ are deterministic, an RNN cannot model uncertainty in the latent states but, on the other hand, the log-likelihood computation is tractable as the integral in (3.2) is straightforward to solve when the transition probabilities are delta functions.

## 4.3 Deep state-space models

In this section we introduce a broad class of non-linear SSMs with Gaussian transitions and that, similarly to VAEs, use deep neural networks to define flexible transition and emission distributions (Krishnan et al., 2015; Fraccaro et al., 2016c; Krishnan et al., 2017). For simplicity in the exposition we will refer to them as *deep state-space models* (DSSM).

In a DSSM the transition distribution is a Gaussian, i.e. $p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t; \boldsymbol{\mu}_t^{(p)}, \mathbf{v}_t^{(p)})$, whose mean and diagonal covariance matrix are a function of the previous latent state $\mathbf{z}_{t-1}$ and current

input $\mathbf{u}_t$ through two deep neural networks:

$$\boldsymbol{\mu}_t^{(p)} = \mathrm{NN}_1^{(p)}(\mathbf{z}_{t-1}, \mathbf{u}_t) \ , \qquad \log \mathbf{v}_t^{(p)} = \mathrm{NN}_2^{(p)}(\mathbf{z}_{t-1}, \mathbf{u}_t) \ . \tag{4.2}$$

If scalability is not an issue, to make the model more general it is also possible to use a Gaussian with full covariance matrix, see (Rezende et al., 2014) for a discussion on possible Gaussian covariance parameterizations. As for VAEs, depending on the type of observations the emission distribution $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$ is typically chosen to be either a Gaussian distribution (real-valued data) or a Bernoulli distribution (binary data). The parameters of both distribution are computed with deep neural networks with input $\mathbf{z}_t$.

The exact parametrization of transition and emission probabilities is problem dependent. For the transitions, the simplest parameterization concatenates $[\mathbf{z}_{t-1}, \mathbf{u}_t]$ and passes this vector through a neural network that returns the mean and the diagonal covariance of the prior over $\mathbf{z}_t$. However, if for example we are doing video modelling and $\mathbf{u}_t = \mathbf{x}_{t-1}$ is an image, it is typically convenient to first pass $\mathbf{u}_t$ through a (convolutional) neural network that does feature extractor, and then concatenate the resulting vector with $\mathbf{z}_{t-1}$. Krishnan et al., (2017) use gated transition functions that allow the model to learn to use linear transitions for some latent dimensions and non-linear ones for others. The parameterization for the emission distribution highly depends on the type of observations. Standard deep neural networks are a good default choice, but when dealing with images it is often better to use convolutional architectures.

For notational simplicity we assume that the initial state $\mathbf{z}_0$ is a fixed and known vector (we could otherwise learn it). The joint distribution is then given by

$$\begin{aligned} p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}, \mathbf{z}_0) &= p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) p_\theta(\mathbf{z}_{1:T}|\mathbf{u}_{1:T}, \mathbf{z}_0) \\ &= \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{z}_t) p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) \end{aligned} \tag{4.3}$$

This distribution specifies the generative process of the data, therefore as in VAEs $p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}, \mathbf{z}_0)$ is often referred to as *generative model*.

### 4.3.1 Amortized inference and parameter learning

Thanks to the deep neural networks used in their parameterization, DSSMs are very expressive and can model a wide range of data distributions. However, as discussed in Section 3.5, due to the non-linearities in the model exact inference is not possible. In Section 3.5.1 we discussed several approximate inference techniques, that are however not scalable enough to large datasets and to high dimensional spaces. The usage of neural networks to define non-linear state-space models has also been previously considered (Valpola and Karhunen, 2002; Raiko and Tornio, 2009), that approximate the posterior using a variational inference procedure that scales quadratically with the dimensionality of the observations, and is therefore not suitable for the large-scale applications we are interested in.

Below we will extend the amortized inference ideas used for VAEs in Section 2.4.2 to the temporal setting, that will allow us to specify a powerful and scalable way to perform joint inference and parameter learning in DSSMs.

#### 4.3.1.1 ELBO derivation

As done for the derivation of the ELBO in Section 2.3, we can introduce a variational approximation conditioned on all quantities that are known at inference time, i.e. $q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)$, and compute the ELBO for a DSSM as:

$$\begin{aligned}
\log p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T}, \mathbf{z}_0) &= \log \int p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}, \mathbf{z}_0) \mathrm{d}\mathbf{z}_{1:T} \\
&= \log \int \frac{p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}, \mathbf{z}_0)}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)} q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0) \mathrm{d}\mathbf{z}_{1:T} \\
&= \log \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)} \left[ \frac{p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}, \mathbf{z}_0)}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)} \right] \\
&\geq \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)} \left[ \log \frac{p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}, \mathbf{z}_0)}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)} \right] = \mathcal{F}_i(\theta, \phi) \ . \qquad (4.4)
\end{aligned}$$

We now have to define a parameterization for the variational distribution $q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)$. If all the sequences had the same length, we could in principle define a deep neural network that outputs the mean and variances of the posterior approximation of the latent variables at all $T$ time steps. However, in many applications the sequences may have different lengths and, more importantly, this parameterization does not exploit the dependencies induced by the temporal structure of the problem. As shown below, we can instead define a variational approximation inspired by the sequential factorization of the true posterior distribution.

Using the independence properties given by the Markovian structure of the model that were discussed in Section 3.1, we can factorize the true intractable posterior distribution as

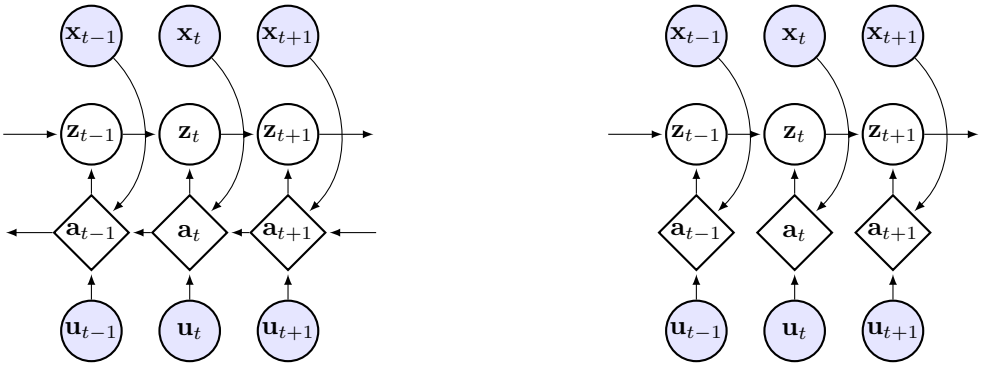
$$\begin{aligned}
p_\theta(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0) &= \prod_{t=1}^{T} p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:T}, \mathbf{u}_{1:T}) \\
&= \prod_{t=1}^{T} p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{t:T}, \mathbf{u}_{t:T}) \ . \qquad (4.5)
\end{aligned}$$

This equation implies that if we know $\mathbf{z}_{t-1}$, then the posterior over $\mathbf{z}_t$ does not depend on past inputs and outputs, but only on present and future ones. The state $\mathbf{z}_{t-1}$ in fact captures all the relevant information coming from the past. We can then approximate the true posterior with a *structured* variational approximation that mimics the factorization in (4.5):

$$q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0) = \prod_{t=1}^{T} q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{t:T}, \mathbf{u}_{t:T}) \ . \qquad (4.6)$$

This approximation to the smoothed posterior distribution shares the same parameters $\phi$ at each time step, and this allows us to handle sequences of variable length. Similarly to VAEs, we can make this inference procedure scalable by defining $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{t:T}, \mathbf{u}_{t:T})$ as an inference network that returns the parameters of a Gaussian distribution. We will discuss the exact parameterization in detail in Section 4.3.1.2.

Since both the joint distribution in (4.3) and the posterior approximation in (4.6) factorize over




***(a)*** *Smoothing* ***(b)*** *Filtering*

***Figure 4.2:*** *Inference networks for smoothing and filtering in a deep state-space model.*

time we can decompose the ELBO as a sum over $T$ terms:

$$\begin{aligned}
\mathcal{F}_i(\theta,\phi) &= \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T},\mathbf{u}_{1:T},\mathbf{z}_0)}\left[\log \frac{p_\theta(\mathbf{x}_{1:T},\mathbf{z}_{1:T}|\mathbf{u}_{1:T},\mathbf{z}_0)}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T},\mathbf{u}_{1:T},\mathbf{z}_0)}\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T},\mathbf{u}_{1:T},\mathbf{z}_0)}\left[\sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t)p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{u}_t)}{q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x}_{t:T},\mathbf{u}_{t:T})}\right] \\
&= \sum_{t=1}^{T} \mathbb{E}_{q_\phi^*(\mathbf{z}_{t-1})}\Big[\mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x}_{t:T},\mathbf{u}_{t:T})}\big[\log p_\theta(\mathbf{x}_t|\mathbf{z}_t)\big] + \\
&\qquad - \mathrm{KL}\Big(q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x}_{t:T},\mathbf{u}_{t:T}) \,\Big\|\, p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{u}_t)\Big)\Big] \,, \qquad (4.7)
\end{aligned}$$

where $q_\phi^*(\mathbf{z}_{t-1})$ denotes the marginal distribution of $\mathbf{z}_{t-1}$ in the variational approximation to the posterior $q_\phi(\mathbf{z}_{1:t-1}|\mathbf{x}_{1:T},\mathbf{u}_{1:T},\mathbf{z}_0)$, given by

$$q_\phi^*(\mathbf{z}_{t-1}) = \int q_\phi(\mathbf{z}_{1:t-1}|\mathbf{x}_{1:T},\mathbf{u}_{1:T},\mathbf{z}_0)\,\mathrm{d}\mathbf{z}_{1:t-2} = \mathbb{E}_{q_\phi^*(\mathbf{z}_{t-2})}\Big[q_\phi(\mathbf{z}_{t-1}|\mathbf{z}_{t-2},\mathbf{x}_{t-1:T},\mathbf{u}_{t-1:T})\Big] \,.$$

Interestingly, we can interpret the ELBO in (4.7) as having a VAE at each time step with a time-varying prior that depends on the previous state. Notice in particular the decomposition of each term in the summation in a reconstruction and regularization term.

While the KL term can be computed analytically, the expectations in the ELBO are still intractable, and we approximate them by sampling from the variational approximation and using Monte Carlo integration as done for VAEs. To reduce the computational cost, it is common to use a single sample at each time step. The parameters $\theta$ of the DSSM and $\phi$ of the inference network can be learned *jointly* by maximizing the ELBO using stochastic gradient ascent, using the reparameterization trick to reduce the variance of the gradients. Recall in particular that this scalable gradient-based optimization is only possible since all the distributions involved in the ELBO computation are parameterized by *differentiable* deep neural networks.

#### 4.3.1.2 Parameterization of the inference network

In this section we will introduce the parameterization for $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{t:T}, \mathbf{u}_{t:T})$ used in (Fraccaro et al., 2016c, Chapter 5), see (Archer et al., 2015; Krishnan et al., 2015; Gao et al., 2016; Krishnan et al., 2017) for alternative but related ones.

When deciding the structure of $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{t:T}, \mathbf{u}_{t:T})$, the main challenge to solve is the dependence of this distribution on a variable number of inputs and outputs at each time step $t$, i.e. $\mathbf{x}_{t:T}$ and $\mathbf{u}_{t:T}$. We approximate this dependence of $\mathbf{z}_t$ on future inputs and outputs in the inference network by introducing an *auxiliary deterministic state* $\mathbf{a}_t$ at each time step that belongs to an RNN running backwards in time. We initialize the hidden state of the backward-recursive RNN as $\mathbf{a}_{T+1} = \mathbf{0}$, and recursively compute

$$\mathbf{a}_t = g_\phi(\mathbf{a}_{t+1}, [\mathbf{u}_t, \mathbf{x}_t]) \ , \tag{4.8}$$

using therefore as input to the RNN the concatenation of the present input and output. The function $g_\phi$ can be for example an LSTM or GRU cell. The variational approximation then becomes $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{t:T}, \mathbf{u}_{t:T}) = q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t)$, see Figure 4.2a for a graphical representation. We notice that the state $\mathbf{z}_t$ directly depends on $\mathbf{z}_{t-1}$ and $\mathbf{a}_t$. The direct dependence of $\mathbf{z}_t$ on $\mathbf{z}_{t-1}$ in the variational approximation is used to encode the information coming from the past. The dependence of $\mathbf{z}_t$ on the present and future inputs and outputs is then encoded in $\mathbf{a}_t$, as in (4.8) the concatenation $[\mathbf{x}_t, \mathbf{u}_t]$ contains the information coming from the present, while the hidden state $\mathbf{a}_{t+1}$ encodes the information coming from the future. Similarly to a VAE, we assume that $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t)$ is a Gaussian distribution whose mean and log-variance are parameterized as

$$\boldsymbol{\mu}_t^{(q)} = \mathrm{NN}_1^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t) \ , \qquad \log \mathbf{v}_t^{(q)} = \mathrm{NN}_2^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t) \ . \tag{4.9}$$

Instead of smoothing, we can do filtering simply by replacing the RNN running backwards in time with a neural network that receives as input the present input and output information, i.e. $\mathbf{a}_t = \mathrm{NN}^{(a)}(\mathbf{x}_t, \mathbf{u}_t)$, see Figure 4.2b.

### 4.3.2 Tightening the bound with particle filters

In Section 2.5.1 we have seen that IWAEs extend VAEs by using importance sampling to define a tighter bound to the log-likelihood than the ELB0. The IWAE bound can be applied of course also to DSSMs, but as we will see below we can do even better than that for problems with a sequential structure. In Section 3.5.1.4 we have presented particle filters as an extension of importance sampling to the temporal setting in which we use a resampling step to make sure that the particles are concentrated in regions of high posterior density. For sequential models like DSSM we can then improve the tightness of the bound by extending IWAEs ideas using particle filters instead of importance sampling. This method was recently introduced independently by three different research groups (Maddison et al., 2017a; Le et al., 2018; Naesseth et al., 2018).

In (3.20) we have seen that as a byproduct of particle filtering we obtain an estimator to the marginal likelihood:

$$\widehat{p}_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T}) = \prod_{t=1}^{T} \frac{1}{R} \sum_{r=1}^{R} w(\mathbf{z}_{1:t}^{(r)}) \ . \tag{4.10}$$

As this quantity is unbiased, we can calculate $p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T})$ by taking the expectation of this estimator with respect to the importance distribution $q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)$. From this, the new bound can be obtained following the derivation of the ELBO in Section 2.3. Notice in particular that the form of the unnormalized weights in (4.10), $w(\mathbf{z}_{1:t}) = \frac{p_\theta(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}|\mathbf{u}_{1:t})}{q(\mathbf{z}_{1:t}|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})}$, is exactly as the term inside the log in (4.4), and this bound reduces therefore to the standard ELBO of a DSSM if we are only considering one particle. The importance distribution $q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0)$, that plays the same role as the variational approximation in this case, can be defined recursively with an inference network $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_t, \mathbf{u}_t)$ parameterized by deep neural networks, that approximates the optimal importance distribution $p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_t, \mathbf{u}_t)$ in (3.19). We can then learn the parameters $\phi$ of this distribution together with the parameters $\theta$ of the DSSM using stochastic gradient ascent, as discussed in Section 4.3.1.

We have seen in Section 3.5.1.4 that it is convenient to only perform the resampling step when the effective sample size (ESS) is lower than a certain threshold. The ESS however is calculated using the weights, that depend in turn on the parameters $\theta$ and $\phi$. This implies that when we compute gradients with respect to these parameters we will have gradient terms that come from the resampling step. Empirically (Maddison et al., 2017a; Le et al., 2018; Naesseth et al., 2018) have shown that these gradients have very high variance, and propose to discard them during training. This introduces small biases in the gradient estimation, but despite this it allows to obtain convincing improvements in terms of final log-likelihood estimation compared to using the ELBO or the IWAE bound (obtained using the method presented in this section with no resampling step).

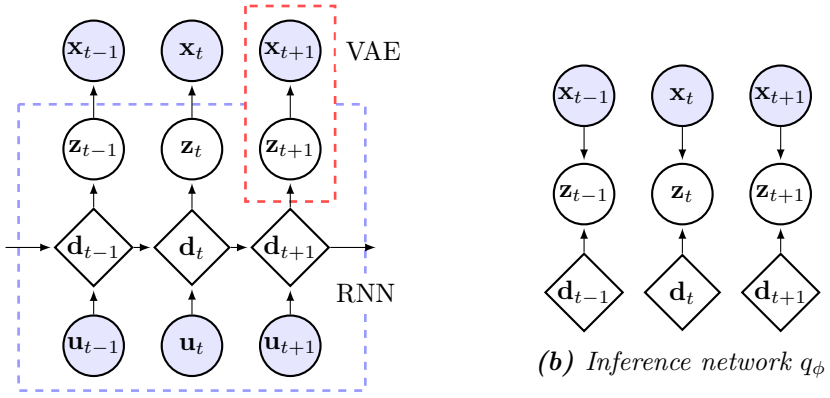
Instead of using sequential Monte Carlo methods (particle filters) to define a tighter bound than the ELBO, Gu et al., (2015) use them to directly approximate the log-likelihood of a SSM, and learn the importance distribution by minimizing the KL diverge from the posterior's sample-based approximation to the importance distribution (i.e. the opposite KL than the one used in variational methods).

## 4.4 Sequential extensions of variational auto-encoders

### 4.4.1 The VAE-RNN model

The DSSM model introduced in Section 4.3 can be seen as having a VAE at each time step with a time-varying prior, so that each latent variable $\mathbf{z}_t$ directly depends on the previous one. An alternative way to introduce correlations among latent variables is by expanding the model hierarchically with a new set of temporally correlated deterministic latent variables, the states of a recurrent neural network (RNN). The VAE-RNN model uses the same VAE at each time step, but with a prior for state $\mathbf{z}_t$ that depends on the information on the past of the sequence captured in the RNN by the state $\mathbf{d}_t$, as illustrated in Figure 4.3a. The generative model of the VAE-RNN is described by the joint distribution of the outputs and unobserved variables given the inputs and initial state $\mathbf{d}_0$ of the RNN, i.e.

$$p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) = p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T})p_\theta(\mathbf{z}_{1:T}|\mathbf{d}_{1:T})p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0)$$




***(a)*** *Generative model $p_\theta$*

***(b)*** *Inference network $q_\phi$*

***Figure 4.3:*** *The VAE-RNN model.*

$$= \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{z}_t)p_\theta(\mathbf{z}_t|\mathbf{d}_t)p_\theta(\mathbf{d}_t|\mathbf{d}_{t-1}, \mathbf{u}_t) \ . \tag{4.11}$$

where $\mathbf{d}_{1:T}$ can be computed by repeatedly applying the state update equation of the RNN given the inputs, $\mathbf{d}_t = f_\theta(\mathbf{d}_{t-1}, \mathbf{u}_t)$. We denote as $\widetilde{\mathbf{d}}_{1:T}$ the value assumed by $\mathbf{d}_{1:T}$ after repeatedly applying the RNN transitions $f_\theta$. The deterministic transitions can then be written from a probabilistic point of view using delta functions centered at $\widetilde{\mathbf{d}}_t$, i.e. $p_\theta(\mathbf{d}_t|\mathbf{d}_{t-1}, \mathbf{u}_t) = \delta(\mathbf{d}_t - \widetilde{\mathbf{d}}_t)$. For simplicity, we assume that the initial state of the RNN is set to $\mathbf{0}$ (we could otherwise learn it). $p_\theta(\mathbf{z}_t|\mathbf{d}_t)$ is typically a Gaussian distribution, whose mean and variance depend on $\mathbf{d}_t$ through deep networks. Notice in particular that given $\mathbf{d}_{1:T}$ the VAE states $\mathbf{z}_{1:T}$ are independent, i.e.

$$p_\theta(\mathbf{z}_{1:T}|\mathbf{d}_{1:T}) = \prod_{t=1}^{T} p_\theta(\mathbf{z}_t|\mathbf{d}_t) \ . \tag{4.12}$$

We can optimize the parameters of the model by extending to the sequential setting the ELBO of the VAE presented in Section 2.4.3, as done for the DSSM in Section 4.3.1. Using Jensen's inequality we can obtain a lower bound for the evidence $\log p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0)$:

$$\begin{aligned}
\log p_\theta(\mathbf{x}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) &= \log \int p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) \mathrm{d}\mathbf{z}_{1:T} \, \mathrm{d}\mathbf{d}_{1:T} \\
&= \log \int \frac{p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0)}{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)} q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0) \mathrm{d}\mathbf{z}_{1:T} \, \mathrm{d}\mathbf{d}_{1:T} \\
&= \log \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)} \left[ \frac{p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0)}{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)} \right] \\
&\geq \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)} \left[ \log \frac{p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0)}{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)} \right] = \mathcal{F}_i(\theta, \phi) \ ,
\end{aligned} \tag{4.13}$$

where we have introduced a variational approximation $q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)$ over all the latent variables of the model, conditioned on the data and the inputs.

Due to the deterministic nature of the RNN, the true posterior over $\mathbf{d}_{1:T}$ coincides with its prior. Furthermore, using the *d-separation* properties (Geiger et al., 1990) of the graphical model in

Figure 4.3a it is easy to show that, conditioned on $\mathbf{d}_{1:T}$, the latent variables of the VAEs at each time step are independent. The *true* posterior distribution $p_\theta(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)$ then factorizes as

$$\begin{aligned} p_\theta(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0) &= p_\theta(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{d}_{1:T}) p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) \\ &= \left( \prod_{t=1}^{T} p_\theta(\mathbf{z}_t|\mathbf{x}_t, \mathbf{d}_t) \right) p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) \ . \end{aligned} \tag{4.14}$$

The variational distribution is an approximation to the true posterior distribution, so we define it to mimic the factorization in (4.14):

$$\begin{aligned} q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0) &= q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{d}_{1:T}) p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) \\ &= \left( \prod_{t=1}^{T} q_\phi(\mathbf{z}_t|\mathbf{x}_t, \mathbf{d}_t) \right) p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) \ . \end{aligned} \tag{4.15}$$

In (4.15) we have introduced the VAE inference network $q_\phi(\mathbf{z}_t|\mathbf{x}_t, \mathbf{d}_t)$ that, unlike the inference network used in the static case (Section 2.4.2), now also depends on the RNN state $\mathbf{d}_t$. A graphical representation can be found in Figure 4.3b. Similarly to the inference network introduced in Section 2.4.2, we can define $q_\phi(\mathbf{z}_t|\mathbf{x}_t, \mathbf{d}_t)$ as a Gaussian distribution whose mean and diagonal covariance depend on the concatenation of $\mathbf{x}_t$ and $\mathbf{d}_t$ through a deep network parameterized by $\phi$. Using (4.11) and (4.15) in (4.13), the prior $p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0)$ cancels out, and the ELBO becomes

$$\begin{aligned} \mathcal{F}_i(\theta, \phi) &= \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)} \left[ \log \frac{p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) p_\theta(\mathbf{z}_{1:T}|\mathbf{d}_{1:T})}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{d}_{1:T})} \right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \widetilde{\mathbf{d}}_{1:T})} \left[ \log \frac{p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) p_\theta(\mathbf{z}_{1:T}|\widetilde{\mathbf{d}}_{1:T})}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \widetilde{\mathbf{d}}_{1:T})} \right] \\ &= \sum_{t=1}^{T} \mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{x}_t, \widetilde{\mathbf{d}}_t)} \left[ \log \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t) p_\theta(\mathbf{z}_t|\widetilde{\mathbf{d}}_t)}{q_\phi(\mathbf{z}_t|\mathbf{x}_t, \widetilde{\mathbf{d}}_t)} \right] \\ &= \sum_{t=1}^{T} \left( \mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{x}_t, \widetilde{\mathbf{d}}_t)} \left[ \log p_\theta(\mathbf{x}_t|\mathbf{z}_t) \right] - \mathrm{KL} \left[ q_\phi(\mathbf{z}_t|\mathbf{x}_t, \widetilde{\mathbf{d}}_t) || p_\theta(\mathbf{z}_t|\widetilde{\mathbf{d}}_t) \right] \right) \end{aligned}$$

After computing $\widetilde{\mathbf{d}}_{1:T}$, we can then rewrite the ELBO as the one obtained if we had $T$ independent data points. As RNNs are differentiable the gradients needed to jointly learn the parameters of the model and the inference network can still be computed using back-propagation and the reparameterization trick as for the VAE in Section 2.4.3. The VAE-RNN model will be used as a baseline when modelling simple videos of moving objects in Section 6.5.

### 4.4.2 Variational recurrent neural networks

In the previous section we have shown that RNNs can be used to extend VAEs to handle sequential data by defining a hierarchical time-varying prior. The VAE-RNN model can however be also understood the other way around, as using VAEs to extend RNNs to handle more complex data distribution. Standard RNNs typically use in fact Gaussian output distributions (or mixture of Gaussians), and as such they may struggle with highly multimodal data distributions. Adding

a VAE to the output of the RNN we can define a very flexible architecture that is suitable to model a wide range of complex data distributions. The model presented in this section is better understood using this latter interpretation of the VAE-RNN model.

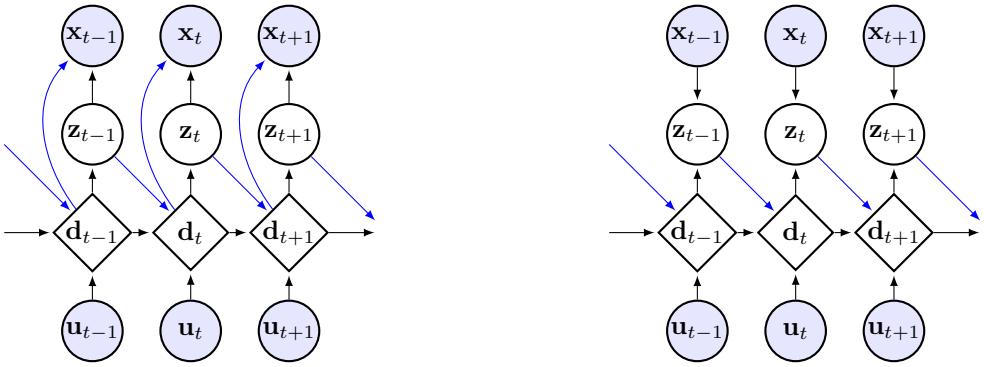
With respect to the standard RNN architecture in Figure 4.1b, we see that in the VAE-RNN model of Figure 4.3a there is no direct dependency from the deterministic hidden states of the RNN $\mathbf{d}_t$ to the outputs $\mathbf{x}_t$. A natural first extension to the VAE-RNN model is therefore the addition of this missing connection, so that the likelihood of the model becomes $p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t)$ instead of $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$. This seemingly minor change is very relevant from the modelling point of view: if $\mathbf{x}_t$ does not directly depend on $\mathbf{d}_t$, $\mathbf{z}_t$ has to encode all the relevant information on the past of the sequence that is captured by $\mathbf{d}_t$. If $\mathbf{x}_t$ is connected to both $\mathbf{d}_t$ and $\mathbf{z}_t$ on the other hand, the model can use the two sets of latent variables to separately encode in each of them the different aspects of the data they are better at modelling. In particular, the *deterministic* states $\mathbf{d}_t$ can be used to capture the overall structure of the data using the flexibility and the power of RNN architectures, while the *stochastic* latent variables $\mathbf{z}_t$ are well suited to model the variability in the data. Notice that typically the dimensionality of $\mathbf{d}_t$ needs to be much higher that the one of $\mathbf{z}_t$ (e.g. 1000 vs 100 dimensions). Being able to capture both aspects is fundamental for example when modelling speech. All speech waveforms share a common structure that follows from the rules of spoken languages, and contain very complex long-term dependencies across time steps. Every person has however its own way to speak, meaning that there is a lot of variability in how different people - and even in how the same person but under different circumstances - pronounce the same sentence. While RNNs are ideal to model the high level structure of the waveforms, as its hidden states are inherently deterministic they would struggle to also model at the same time all the nuances and variations across different speakers in the data. For this, extending the deterministic state $\mathbf{d}_t$ with a stochastic component $\mathbf{z}_t$ that is able to naturally encode the variability in the data is fundamental.

The speech modelling example can be also used to justify another possible extension to the VAE-RNN model. As we said, the stochastic latent variables $\mathbf{z}_t$ model the variability in the data, for example due to the particular vocal characteristics of the speaker. For a given sequence it is then reasonable to assume that the variability is consistent over time, i.e. that each latent variable $\mathbf{z}_t$ is directly affected by the value of $\mathbf{z}_{1:t-1}$. This is however not the case in the generative model of the VAE-RNN in Figure 4.3a, where to obtain the prior $p_\theta(\mathbf{z}_t|\mathbf{d}_t)$ we only need to compute the RNN states $\mathbf{d}_{1:t}$ given the inputs, and not any of the previous states $\mathbf{z}_{1:t-1}$. Below we will then add an arrow from $\mathbf{z}_{t-1}$ to $\mathbf{d}_t$, so that $\mathbf{z}_t$ will depend on $\mathbf{z}_{t-1}$ indirectly through $\mathbf{d}_t = f_\theta(\mathbf{d}_{t-1}, \mathbf{z}_{t-1}, \mathbf{u}_t)$.

The *variational recurrent neural network (VRNN)* (Chung et al., 2015) can be seen as an extension of the VAE-RNN model where we add the additional dependencies of $\mathbf{x}_t$ on $\mathbf{d}_t$ and of $\mathbf{d}_t$ on $\mathbf{z}_{t-1}$ as discussed above and depicted in Figure 4.4a.[1] The generative model is then defined by the joint probability

$$\begin{aligned} p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0) &= p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}, \mathbf{d}_{1:T}) p_\theta(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0) \\ &= \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t) p_\theta(\mathbf{z}_t|\mathbf{d}_t) p_\theta(\mathbf{d}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t-1}, \mathbf{u}_t) \ . \end{aligned} \tag{4.16}$$

[1] To be consistent with the notation used throughout the whole thesis, we have changed the name of some variables with respect to the original paper of Chung et al., (2015). Also, we have generalized the model to have inputs $\mathbf{u}_t$ instead of only presenting the special case $\mathbf{u}_t = \mathbf{x}_{t-1}$ as in the original paper.




*(a) Generative model $p_\theta$* *(b) Inference network $q_\phi$*

***Figure 4.4:*** *Variational recurrent neural network (VRNN). The blue arrows are the additional dependencies introduced with respect to the VAE-RNN model of Figure 4.3.*

Similarly to the VAE-RNN model, to take into account the fact that the RNN states are deterministic we define $p_\theta(\mathbf{d}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t-1}, \mathbf{u}_t)$ as a delta function whose center is computed with the recursion $\mathbf{d}_t = f_\theta(\mathbf{d}_{t-1}, \mathbf{z}_{t-1}, \mathbf{u}_t)$.

The posterior approximation chosen in (Chung et al., 2015) can be written as

$$q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0) = \prod_{t=1}^{T} q_\phi(\mathbf{z}_t|\mathbf{x}_t, \mathbf{d}_t) p_\theta(\mathbf{d}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t-1}, \mathbf{u}_t) \ . \tag{4.17}$$

Recalling that from Bayes' rule we know that the true posterior will be proportional to the joint distribution, i.e. $p_\theta(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0) \propto p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0)$, we see that in the VRNN we introduce the inference network $q_\phi(\mathbf{z}_t|\mathbf{x}_t, \mathbf{d}_t)$ to approximate the unnormalized factor $p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t)p_\theta(\mathbf{z}_t|\mathbf{d}_t)$ in (4.16). The specific form of $q_\phi(\mathbf{z}_t|\mathbf{x}_t, \mathbf{d}_t)$ is typically chosen as in the VAE-RNN as a Gaussian whose mean and variance depend on $\mathbf{x}_t$ and $\mathbf{d}_t$ through deep networks. Now that we have defined both the joint and the variational approximation, we can compute the ELBO as

$$\begin{aligned}
\mathcal{F}_i(\theta, \phi) &= \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0)} \left[ \log \frac{p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0)}{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0)} \right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0)} \left[ \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t) p_\theta(\mathbf{z}_t|\mathbf{d}_t))}{q_\phi(\mathbf{z}_t|\mathbf{x}_t, \mathbf{d}_t)} \right] .
\end{aligned}$$

Unlike the VAE-RNN model, where samples from the variational approximation could be computed in parallel given the deterministic units, due to the connection from $\mathbf{z}_{t-1}$ to $\mathbf{d}_t$ in the VRNN the sample at time $t$ will depend on the past samples as well. Samples from $q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0)$ can be easily obtained sequentially by ancestral sampling as illustrated in Algorithm 1. Tighter bounds for the VRNN can be obtained using particles filters as discussed in Section 4.3.2, see (Maddison et al., 2017a) for an example of application. We will use the VRNN as a baseline in the speech modelling experiments of Section 5.4.

We finally notice that the VRNN can be seen as a DSSM in which the state $\mathbf{s}_t$ is split in a stochastic and in a deterministic component, i.e. $\mathbf{s}_t = [\mathbf{z}_t, \mathbf{d}_t]$. A graphical representation can be

**Algorithm 1** Sampling procedure for the Gaussian posterior approximation $q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0)$ of a VRNN.

**inputs**: $\mathbf{u}_{1:T}$ and $\mathbf{x}_{1:T}$
initialize $\mathbf{d}_0$ and $\mathbf{z}_0$
**for** $t = 1$ to $T$ **do**
  $\mathbf{d}_t = f_\theta(\mathbf{d}_{t-1}, \mathbf{z}_{t-1}, \mathbf{u}_t)$ % RNN recursion
  $\boldsymbol{\mu}_t^{(q)} = \mathrm{NN}_1^{(q)}(\mathbf{d}_t, \mathbf{x}_t)$ % Posterior mean
  $\log \mathbf{v}_t^{(q)} = \mathrm{NN}_2^{(q)}(\mathbf{d}_t, \mathbf{x}_t)$ % Posterior log-variance
  $\mathbf{z}_t \sim \mathcal{N}(\mathbf{z}_t; \boldsymbol{\mu}_t^{(q)}, \mathbf{v}_t^{(q)})$ % Posterior sample
**end for**

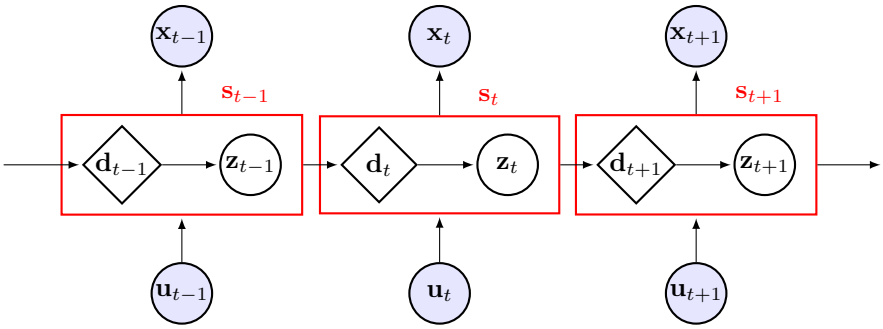



***Figure 4.5:*** *VRNN in state-space form.*

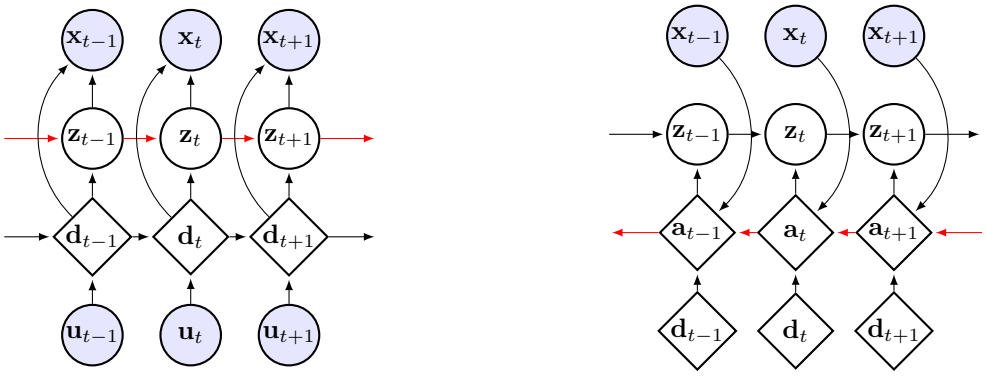
found in Figure 4.5. The transition distribution is assumed to factorize in the following way:

$$p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{u}_t) = p_\theta(\mathbf{z}_t, \mathbf{d}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t-1}, \mathbf{u}_t) = p_\theta(\mathbf{z}_t|\mathbf{d}_t)p_\theta(\mathbf{d}_t|\underbrace{\mathbf{z}_i^{t-1}, \mathbf{d}_i^{t-1}}_{\mathbf{s}_i^{t-1}}, \mathbf{u}_t) \ ,$$

so that the dependence on the past and inputs is only captured by the deterministic component, that is then used to condition the stochastic one. The emission distribution depends on both components, i.e. $p_\theta(\mathbf{x}_t|\mathbf{s}_t) = p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t)$, and this leads to a joint distribution that coincides with (4.16):

$$\begin{aligned} p_\theta(\mathbf{x}_{1:T}, \mathbf{s}_{1:T}|\mathbf{s}_{1:T}, \mathbf{s}_0) &= \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{s}_t)p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{u}_t) \\ &= \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t)p_\theta(\mathbf{z}_t|\mathbf{d}_t)p_\theta(\mathbf{d}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t-1}, \mathbf{u}_t) \ . \end{aligned}$$

In a VRNN, the state $\mathbf{z}_t$ depends on $\mathbf{z}_{t-1}$ indirectly through the deterministic RNN state $\mathbf{d}_t$. The RNN states represent therefore a deterministic bottleneck through which all the stochasticity has to pass, making it difficult for the VRNN to properly model how the uncertainty in the latent variables propagates across time steps. As shown in the next section, this can be done by making $\mathbf{z}_t$ directly depend on $\mathbf{z}_{t-1}$ as in a DSSM, which is also a more natural way to model temporal correlations among latent states. While this allows us to define a more expressive model than the VRNN, this dependency makes inference harder, as we now have to perform filtering and smoothing over a chain of latent variables as in Section 4.3.1.




***(a)*** *Generative model $p_\theta$* ***(b)*** *Inference network $q_\phi$*

***Figure 4.6:*** *A stochastic recurrent neural network. The red arrows denote the different dependencies introduced with respect to the VRNN model of Figure 4.4.*

## 4.5 Stochastic recurrent neural networks

*Stochastic recurrent neural networks are introduced in detail in the research paper in Chapter 5. Here we will only briefly summarize the main ideas behind this model, mostly focusing on how it relates to the other models presented in this chapter.*

*Stochastic recurrent neural networks* (SRNN) (Fraccaro et al., 2016c) are formed by staking an RNN and a DSSM, see Figure 4.6a for a graphical representation. Instead of the arrow from $\mathbf{z}_{t-1}$ to $\mathbf{d}_t$ as in the VRNN model of Figure 4.4a, in a SRNN $\mathbf{z}_{t-1}$ is directly connected to $\mathbf{z}_t$. SRNNs combine the advantages of RNNs and DSSMs in a principled way: the RNN can be used to capture complex long-term dependencies in its deterministic states, while the DSSM can model uncertainty in the latent representation through the stochastic states. The generative model of the SRNN model is given by

$$\begin{aligned} p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T} | \mathbf{u}_{1:T}, \mathbf{d}_0, \mathbf{z}_0) &= p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}, \mathbf{d}_{1:T}) p_\theta(\mathbf{z}_{1:T}|\mathbf{d}_{1:T}, \mathbf{z}_0) p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) \\ &= \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t) p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{d}_t) p_\theta(\mathbf{d}_t|\mathbf{d}_{t-1}, \mathbf{u}_t) \ . \end{aligned} \tag{4.18}$$

We notice in particular that the transition density of the DSSM now depends on the deterministic states of the RNN, i.e. $p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{d}_t) = \mathcal{N}(\mathbf{z}_t; \boldsymbol{\mu}_t^{(p)}, \mathbf{v}_t^{(p)})$ with

$$\boldsymbol{\mu}_t^{(p)} = \mathrm{NN}_1^{(p)}(\mathbf{z}_{t-1}, \mathbf{d}_t) \ , \qquad \log \mathbf{v}_t^{(p)} = \mathrm{NN}_2^{(p)}(\mathbf{z}_{t-1}, \mathbf{d}_t) \ . \tag{4.19}$$

The DSSM can therefore utilize the long-term information that is captured by the RNN. Furthermore, in the SRNN the RNN transitions $p_\theta(\mathbf{d}_t|\mathbf{d}_{t-1}, \mathbf{u}_t)$ are entirely deterministic over time since they no longer depend on the noisy samples of $\mathbf{z}_{t-1}$ as in the VRNN.

The clear separation between deterministic and stochastic states in a SRNN allows us to perform inference and parameter learning extending the ideas presented in Section 4.3.1 for DSSMs. As the states $\mathbf{d}_{1:T}$ are deterministic and do not depend on the stochastic states their posterior coincides with their prior, and we can therefore write the posterior distribution over the latent variables of the SRNN as:

$$p_\theta(\mathbf{z}_{1:T}, \mathbf{d}_{1:T} | \mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0, \mathbf{d}_0) = p_\theta(\mathbf{z}_{1:T}|\mathbf{d}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0) p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) \ . \tag{4.20}$$

In (4.20) the first term represents the intractable posterior over the states of the DSSM, while the second term represents the prior RNN transition probabilities. A similar factorization was used in (4.14) for the VAE-RNN model. We assume that the variational approximation factorizes similarly to the posterior in (4.20), i.e.

$$q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0, \mathbf{d}_0) = q_\phi(\mathbf{z}_{1:T}|\mathbf{d}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0)p_\theta(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0) \ . \tag{4.21}$$

During inference, the states $\mathbf{d}_{1:T}$ at all time steps can be easily computed given only the inputs of the model with the RNN recursions that define $p_\theta(\mathbf{d}_t|\mathbf{d}_{t-1}, \mathbf{u}_t)$. Once these are known, the SRNN can be seen as a DSSM whose inputs are the RNN states, and we can therefore use the same amortized inference procedure as for the DSSM in Section 4.3.1. In particular, we can use the same inference networks for filtering and smoothing, that were justified using the conditional independence properties of the true posterior distribution given by the Markovian structure of the model. In Figure 4.6b we show the backward-recursive inference network used for smoothing, that allows us to take also into account information coming from the future inputs and outputs when computing the approximate posterior. As for the VRNN, also for the SRNN we can obtain a tighter lower bound using particle filters (Section 4.3.2). As shown in Section 5.4, SRNNs achieve state of the art performances in speech modelling (outperforming VRNNs by a large margin), and perform comparably to related methods for polyphonic music modelling.

A number of works have recently built on the SRNN model presented in this section. (Goyal et al., 2017) extends the VRNN model of Section 4.4.2 with SRNN's backwards-recurring RNN, adding an auxiliary term in the ELBO that forces the latent variables to encode information about the future. This idea if further developed in the Variational Bi-LSTM model of (Shabanian et al., 2017). Finally, (Liu et al., 2017) builds an architecture similar to the SRNN, using however an Hidden Markov Model with discrete random variables instead of a DSSM, that is trained using a continuous relaxation of the discrete variables defined with the Gumbel-Softmax distribution (Maddison et al., 2017b; Jang et al., 2016).

Apart from the the models already presented in this chapter, the extension of RNNs with stochastic units has been also explored in several other works. STORN (Bayer and Osendorfer, 2014) and DRAW (Gregor et al., 2015) use Gaussian stochastic units independent between time steps as input to the deterministic units of an RNN. (Gan et al., 2015) uses a recurrent model with discrete latent units that is optimized using the NVIL algorithm (Mnih and Gregor, 2014). (Zheng et al., 2017) defines a combination of SSMs and LSTMs that is trained with a stochastic EM approach, where the expectation in E-step is approximated using samples from sequential Monte Carlo.

## 4.6 Learning disentangled representations with structured priors

All the models presented so far in this chapter are very flexible architectures that can be used to model a wide range of data distributions. However, to be able to do so they are often defined with deep neural networks with a very high number of parameters, and this implies that (1) they require a lot of data and (2) it is difficult to interpret what they are learning. As we will show below, we can often counteract this data inefficiency and black-box nature by carefully inserting some structure in the model, e.g. using domain knowledge on the task at hand, that helps the model to learn disentangled representations. Each latent variable will then represent a

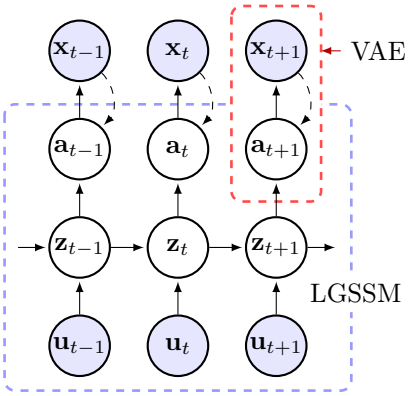



***Figure 4.7:*** *A Kalman variational auto-encoder. Solid arrows represent the generative model while dashed arrows represent the VAE inference network.*

meaningful and interpretable concept, and training will often require less data as this additional prior knowledge encoded in the structure of the model will simplify the learning task.

In Section 4.6.1 we will focus on unsupervised learning of disentangled visual and dynamics representations in the sequential setting by carefully designing a model that combines LGSSMs and VAEs. We will then briefly show in Section 4.6.2 that we can use probabilistic graphical models to introduce structure to the model. Recent works have also focused on using similar ideas in the static setting, e.g. in (Higgins et al., 2017a; Deng et al., 2017). In particular, Higgins et al., (2017a) learn to disentangle representations in a standard VAE by modifying the ELBO rather than designing the model with some structure in it.

### 4.6.1 Kalman variational auto-encoders

*Kalman variational auto-encoders are introduced in detail in the research paper in Chapter 6. Here we will only briefly summarize the main ideas behind this model, mostly focusing on how it relates to the other models presented in this chapter.*

In Section 3.4 we have considered the experiment of tracking a ball given noisy observations of its $(x, y)$ coordinates. We now assume that instead of these 2-dimensional observations we observe at each time step a 32x32 image $\mathbf{x}_t$ of the ball flying in vacuum, as illustrated in Figure 3.6. We then want to model videos formed by 1024-dimensional frames. In Section 3.4 we have seen that a LGSSM is ideal to represent the 2-dimensional trajectory of the ball, as we can derive its parameterization from Newton's equations of motion. However, to model the high-dimensional images we need a non-linear emission distribution. Can we exploit the prior knowledge on the physics of flying ball that is encoded in the LGSSM while dealing with high-dimensional observations? As we will see below, we can do this by constructing a model that disentangles visual and dynamic information.

If we compress the high-dimensional information in the image into a 2-dimensional latent variable representing the position of the ball, we can model the trajectories in this learned manifold with a LGSSM and perform filtering, smoothing and missing data imputation as described in Section 3.4. This is the key intuition behind a *Kalman variational auto-encoder* (KVAE) (Fraccaro et al.,

2017), that factorizes the latent structure at each time step in a visual and dynamic component as illustrated in Figure 4.7. Similarly to the VAE-RNN model of Section 4.4.1, a KVAE has a VAE at each time steps, that encodes the high-dimensional observations into a low dimensional latent state $\mathbf{a}_t$ which learns to capture the relevant visual information in the observations. The prior over these latent states is time-varying, and parameterized with a LGSSM with states $\mathbf{z}_t$ that models the temporal dynamics of the system. As shown below, we can learn jointly the parameters of both the VAE and the LGSSM. Notice that by working in this lower dimensional manifold instead of in image-space, we also avoid the computational issues due to the cubic scaling of the Kalman filter with the output dimensionality discussed in Section 3.4.1.

In the ball tracking experiment we set for example $\mathbf{a}_t \in \mathbb{R}^2$ and $\mathbf{z}_t \in \mathbb{R}^4$. By constraining the dimensionality of these vectors we force the model to learn to use $\mathbf{a}_t$ to model the noisy position of the ball in the frame (in a space that is possibly rotated and scaled w.r.t the $(x, y)$ plane in Figure 3.2), and $\mathbf{z}_t$ to model the position and velocity of the ball in this new space similarly to the example in Section 3.4. While we do not explicitly tell the model to use these latent variables in this way, the model will learn to do it as it is the only way to maximize the ELBO introduced below in (4.27) with such low-dimensional latent states. We could of course also use much higher dimensional latent spaces, but training the KVAE would become more difficult, computationally expensive and require more data.

More in detail, we define the LGSSM with parameters $\gamma = [\gamma_1, .., \gamma_T]$ as in Section 3.4:

$$\begin{aligned} p_\gamma(\mathbf{a}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}) &= p_\gamma(\mathbf{a}_{1:T}|\mathbf{z}_{1:T})\, p_\gamma(\mathbf{z}_{1:T}|\mathbf{u}_{1:T}) \\ &= \prod_{t=1}^{T} p_{\gamma_t}(\mathbf{a}_t|\mathbf{z}_t) \cdot p(\mathbf{z}_1) \prod_{t=2}^{T} p_{\gamma_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)\ , \end{aligned} \tag{4.22}$$

where the transition and emission distributions are respectively

$$p_{\gamma_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t; \mathbf{A}_t\mathbf{z}_{t-1} + \mathbf{B}_t\mathbf{u}_t, \mathbf{Q}_t) \tag{4.23}$$

$$p_{\gamma_t}(\mathbf{a}_t|\mathbf{z}_t) = \mathcal{N}(\mathbf{a}_t; \mathbf{C}_t\mathbf{z}_t, \mathbf{R}_t)\ . \tag{4.24}$$

Given the latent outputs of the LGSSM we model the observations with the VAE decoder $p_\theta(\mathbf{x}_t|\mathbf{a}_t)$, i.e. $p_\theta(\mathbf{x}_{1:T}|\mathbf{a}_{1:T}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{a}_t)$. The joint distribution of the KVAE is the product of these distributions:

$$p(\mathbf{x}_{1:T}, \mathbf{a}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}) = p_\theta(\mathbf{x}_{1:T}|\mathbf{a}_{1:T})\, p_\gamma(\mathbf{a}_{1:T}|\mathbf{z}_{1:T})\, p_\gamma(\mathbf{z}_{1:T}|\mathbf{u}_{1:T})\ .$$

We can learn the parameters of the VAE and the LGSSM by introducing a variational approximation $q(\mathbf{a}_{1:T}, \mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T})$ and maximizing the ELBO obtained as usual using Jensen's inequality:

$$\begin{aligned} \log p(\mathbf{x}_{1:T}|\mathbf{u}_{1:T}) &= \log \int p(\mathbf{x}_{1:T}, \mathbf{a}_{1:T}, \mathbf{z}_{1:T}|\mathbf{u}_{1:T}) \\ &\geq \mathbb{E}_{q(\mathbf{a}_{1:T}, \mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T})} \left[\log \frac{p_\theta(\mathbf{x}_{1:T}|\mathbf{a}_{1:T}) p_\gamma(\mathbf{a}_{1:T}|\mathbf{z}_{1:T}) p_\gamma(\mathbf{z}_{1:T}|\mathbf{u}_{1:T})}{q(\mathbf{a}_{1:T}, \mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T})}\right] \\ &= \mathcal{F}(\theta, \gamma, \phi)\ . \end{aligned} \tag{4.25}$$

We choose a variational approximation that allows us to exploit the knowledge we have on the exact posterior of a LGSSM. If we knew the latent variables of the VAE $\mathbf{a}_{1:T}$, we could compute

the conditional posterior over the LGSSM states $p_\gamma(\mathbf{z}_{1:T}|\mathbf{a}_{1:T}, \mathbf{u}_{1:T})$ with the Kalman filtering and smoothing algorithm introduced in Section 3.4.1. We then factorize the variational distribution as

$$\begin{aligned} q(\mathbf{a}_{1:T}, \mathbf{z}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}) &= q_\phi(\mathbf{a}_{1:T}|\mathbf{x}_{1:T})\, p_\gamma(\mathbf{z}_{1:T}|\mathbf{a}_{1:T}, \mathbf{u}_{1:T}) \\ &= \prod_{t=1}^{T} q_\phi(\mathbf{a}_t|\mathbf{x}_t)\, p_\gamma(\mathbf{z}_{1:T}|\mathbf{a}_{1:T}, \mathbf{u}_{1:T}) \ , \end{aligned} \tag{4.26}$$

where $q_\phi(\mathbf{a}_t|\mathbf{x}_t)$ is the VAE inference network. In this way we can easily get samples form the posterior approximation by first sampling $\mathbf{a}_{1:T}^{(s)}$ from the VAE inference network given the observations $\mathbf{x}_{1:T}$, and then sampling from the exact posterior $p_\gamma(\mathbf{z}_{1:T}|\mathbf{a}_{1:T}^{(s)}, \mathbf{u}_{1:T})$. With this choice of the joint and variational distributions the ELBO in (4.25) becomes

$$\begin{aligned} \mathcal{F}(\theta, \gamma, \phi) = \mathbb{E}_{q_\phi(\mathbf{a}_{1:T}|\mathbf{x}_{1:T})} \Bigg[ \log \frac{p_\theta(\mathbf{x}_{1:T}|\mathbf{a}_{1:T})}{q_\phi(\mathbf{a}_{1:T}|\mathbf{x}_{1:T})} + \\ + \mathbb{E}_{p_\gamma(\mathbf{z}_{1:T}|\mathbf{a}_{1:T}, \mathbf{u}_{1:T})} \left[ \log \frac{p_\gamma(\mathbf{a}_{1:T}|\mathbf{z}_{1:T}) p_\gamma(\mathbf{z}_{1:T}|\mathbf{u}_{1:T})}{p_\gamma(\mathbf{z}_{1:T}|\mathbf{a}_{1:T}, \mathbf{u}_{1:T})} \right] \Bigg] \ . \end{aligned} \tag{4.27}$$

We can perform end-to-end training of this model and learn the parameters $\theta$ and $\phi$ of the VAE and $\gamma$ of the LGSSM by jointly maximizing the ELBO. The intractable expectations can be computed with Monte Carlo integration (even with a single sample) and we use the reparameterization trick to obtain low-variance gradients.

Often the dynamics of the objects are non-linear, e.g. when the ball hits a wall, and the LGSSM assumptions no longer hold. In Section 6.3.3 we will show that we can deal with these situations by non-linearly changing the parameters $\gamma_t$ of the LGSSM over time as a function of the latent encodings $\mathbf{a}_{1:t-1}$ up to time $t-1$. Importantly, this allows us to preserve the linear dependency between consecutive states in the LGSSM, and still be able to use the Kalman filtering and smoothing algorithms for posterior inference and missing data imputation. Results on using the KVAE to model the (non-linear) dynamics of the moving ball can be found in Section 6.5.

A recent related work can be found in (Li and Mandt, 2018), that shows how a careful model design can be used to also disentangle a static component in the sequence by extending the model with a time-invariant latent variable. This allows for example to deal with sequences of objects with different shapes and colors.

### 4.6.2 Structured priors with probabilistic graphical models

*Probabilistic graphical models* (PGMs) are a very general framework to represent relationships among random variables by defining joint probability distributions as a product of factors each of which only depends on a subset of the variables, see (Bishop, 2006) for a comprehensive review on the topic. Both latent variable models and state-space models can be seen as special cases of PGMs.

As shown in (Johnson et al., 2016; Lin et al., 2018) we can combine VAEs with PGMs in which we express prior knowledge on the application at hand, as needed when learning disentangled representation and interpretable models. The PGM is used to define a hierarchical prior distribution that helps to extract useful structure from the data (unlike the Gaussian prior of a

VAE). To approximate the intractable posterior over the latent variables of the PGM, we could in principle follow a black-box approach using inference networks as for most of the models presented in this thesis. However, (Johnson et al., 2016; Lin et al., 2018) argue that better posterior approximations can be defined by exploiting the structure of the PGM and existing approximate inference procedures for it, and combine therefore amortized inference using inference networks and Variational Message Passing (Winn and Bishop, 2005). The forward-backward procedure of the Kalman smoother is a message passing algorithm, and the KVAE presented in 4.6.1 can therefore also be seen as a special case of this framework.

## 4.7 Sequential models with external memory architectures

Some applications require models with a particularly high-memory capacity. In reinforcement learning (RL) for example, we may want to build agents that can remember a large number of past experiences and use this knowledge to better plan the sequence of actions needed to solve new tasks. In this cases, using RNNs with memory cells such as LSTMs (Hochreiter and Schmidhuber, 1997) or GRUs (Chung et al., 2014) as in VRNNs and SRNNs is typically not enough. To be able to increase the memory capacity in these architectures we would need in fact to largely increase the dimensionality of the RNN states $\mathbf{d}_t$. However, the scaling of the number of parameters of the RNN is quadratic with respect to the dimensionality of $\mathbf{d}_t$, and this would make learning impracticable both in terms of computational requirements and in terms of the amount of data needed to learn these models. In the static setting, a recent trend in the deep learning community to deal with high memory capacities is to design data structures that work as external memory architectures and are easy to integrate with neural networks (Graves et al., 2014; Sukhbaatar et al., 2015; Miller et al., 2016; Graves et al., 2016; Li et al., 2016; Bornschein et al., 2017). These ideas have been extended to the sequential setting by Gemici et al., (2017), that learn to write/read from external memories using differentiable memory addressing mechanisms.

In Section 4.7.1 we present a generative temporal model for RL agents walking in partially-observed 3D environments, which combines a structured SSM prior similarly to the KVAE of Section 4.6.1 and a non-parametric memory used to store what the agent sees in the environment at each time step.

### 4.7.1 Generative temporal models with spatial memory

*Generative temporal models with spatial memory are introduced in detail in the research paper in Chapter 7. Here we will only briefly summarize the main ideas behind this model, mostly focusing on how it relates to the other models presented in this chapter.*

Imagine to be training an RL agent to walk and explore a large 3D environment for hundreds of steps and successively predict future observations given a specific set of actions that bring it to a previously visited location. The input $\mathbf{u}_t$ to the model at each time step is an action that makes the agent move/rotate in any direction, while the output $\mathbf{x}_t$ is what the agent is observing. This task would be simple to solve if the agent could remember at each time step both its location and the observations. However, the true location is not known, and has to be inferred from the

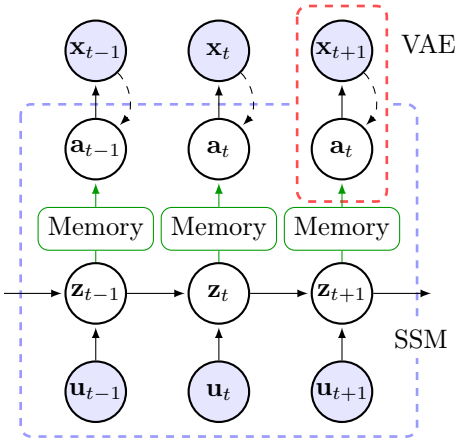



***Figure 4.8:*** *A generative temporal model with spatial memory. Solid arrows represent the generative model while dashed arrows represent the VAE inference network. Green lines represent the additional dependencies with respect to the KVAE model of Figure 4.7.*

sequence of actions and some knowledge on the physics of the moving agent. As explained below, we can solve this task by carefully designing the model with a structured prior as discussed in Section 4.6.

We model the agent dynamics with a SSM with a 3-dimensional state $\mathbf{z}_t$, so that the model can learn to use two components to represent the position of the agent in the environment and the third component to represent in which direction the agent is looking. The transition probabilities can be used to model how each action changes the state of the agent, and its parameters can be learned from data. Also, instead of remembering the full high-dimensional observations, it is easier for the model to remember a compressed representation, that can be obtained for example with a VAE with latent state $\mathbf{a}_t$. The resulting model is the *generative temporal model with spatial memory* (GTM-SM), that is shown in Figure 4.8. We can notice that it is very close to the KVAE introduced in Section 4.6.1, with two important differences:

1. The transitions need to be non-linear, as they need to model the agent's movement that is subject to momentum, friction and saturation. As opposed to the KVAE we are therefore not able to use a LGSSM and leverage its exact inference procedure.

2. While exploring the environment, the agent needs to be able to memorize at each time step its location (the state of the SSM) and the encoded observation (the latent variable from the VAE). These learned and disentangled representations of the environment are stored in a non-parametric spatial memory called *differentiable neural dictionary* (DND) (Pritzel et al., 2017). While predicting a future observation $\mathbf{x}_t$ the agent can then infer its future state $\mathbf{z}_t$ given the new sequence of actions, and retrieve from memory what it had seen before in that location (or close to it). To model this, the emission distribution of the SSM, i.e. the VAE prior, needs to also depend on the memory: $p_\theta(\mathbf{a}_t|\mathbf{z}_t, \text{Memory})$.

This model is presented in detail in Chapter 7, where we can also see that it is able to coherently predict over hundreds of time steps across a range of partially-observed 2D and 3D environments (Section 7.4).

## 4.8 Lessons learned in training sequential deep latent variable models

The implementation of this class of models is fairly simple, thanks to the availability of deep learning libraries that use automatic differentiation to compute the gradients needed during training. However, getting the models to perform well may be much more challenging and require some experience. We therefore share below some lessons learned with a lot of trial an error on how to train deep latent variable models for sequential data.

Most of the challenges faced during training are due to the usage of deep neural networks within the probabilistic model. While their flexibility is beneficial in terms of modelling power, the behavior of complex neural architectures such as the ones presented in this thesis can be difficult to predict. It is then fundamental to understand in depth the role of all the components that form model, inference network and cost function, and how they interact and influence each other. To achieve this it can be beneficial to:

- Start from toy examples the model has to be able to solve. This allows to become more familiar with the working principles of the model, and to detect many of the modelling issues that would be impossible to isolate in more complex applications. Also, toy examples are often very fast to run, and can produce lots of intuitions and visualizations that can be used to assess the performance of the model on more complex tasks (as well as used when writing the paper to give intuition to the reader). The complexity of the toy example can be often increased as we start learning how to train the model, until the point at which the complexity is similar to the one of the original task we want to solve. If the model does not perform well in a basic toy example it is often pointless to try it as it is on more complex tasks.

- During training it is very informative to monitor the evolution of lots of statistics, such as the variance of all the distributions in the model, the various term that form the ELBO and the norm of the gradients. We ideally want one informative statistic for each component of the model, inference network and cost function.

Following these suggestions it is easier to assess if parts of the network are not being used or do not work as we expected, as well as to come up with some tricks that can be used to improve the training procedure. Also, they can give a better idea on how to tune some of the hyperparameters of the model.

This approach is of course more challenging and time consuming than the more traditional empirical deep learning approach in which we try many different complex neural architectures until the performances are good. However, our experience with these models suggests that while in theory big neural networks could learn to automatically do these very complex tasks, in practice they rarely do. In the following we will see that in this case the only way to achieve good performances is to have a deep understanding of the models and of their training dynamics and use it to come up with principled training tricks.

### 4.8.1 Training tricks

We now present some of the issues we found when training some of the architectures presented in this chapter, focusing on how to detect them, providing some intuition on why they appear and listing some of the tricks we found useful to solve them. Coming up with the right training tricks to best learn complex deep learning models has become a fundamental step in a researcher's agenda, as they can make huge differences in terms of final performances. For the deep latent variable models for sequential data presented in this chapter some of the tricks are the same as the ones typically used for VAEs. There is however an additional challenge given by the fact that in this case we are dealing with learned time-varying priors, as opposed to the fixed isotropic Gaussian prior typically used in VAEs.

One of the main issues encountered while training deep latent variable models is their difficulty in learning to fully exploit the stochasticity in the latent states. It is common to notice that in the beginning of training the KL term in the ELBO becomes very small and never recovers. This suggests that in many of the dimensions of the latent variable the variational approximation coincides with the prior, making the corresponding unit essentially inactive (Burda et al., 2015; Sønderby et al., 2016b). From an optimization point of view we can see this as an initially attractive local minimum, as in this case there will be no penalty in the ELBO from the KL term (we are maximizing the ELBO, in which there is a minus in front of the non-negative KL term). In this case a common solution to mitigate this issue is to decrease the importance of minimizing the KL term in the beginning of training by:

- Multiplying the KL term by a constant $\beta$ that is annealed from 0 to 1 during training (Bowman et al., 2015; Sønderby et al., 2016b)
- Modifying the ELBO so that decreasing the KL term below a given threshold is no longer advantageous, as in the *free bits* approach of Kingma et al., (2016).

In temporal models for which we are learning the time-varying prior, small values of the KL term can be also caused by the fact that for the model it may be particularly challenging to learn to approximate the posterior. In particular it may be difficult for the variational approximation to keep track of the changes in the posterior distribution caused by the non-stationary time-varying prior. This is the case for example in DSSMs and SRNNs. An effective solution to make it easier for the model to learn a good variational approximation and therefore exploit the stochasticity is to

- learn only the residual between the mean of the predictive prior distribution $\boldsymbol{\mu}_t^{(p)}$ and the the mean of the variational approximation $\boldsymbol{\mu}_t^{(q)}$ at each time step (Fraccaro et al., 2016c). We then modify (4.9) to
$$\boldsymbol{\mu}_t^{(q)} = \boldsymbol{\mu}_t^{(p)} + \mathrm{NN}_1^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t) \ .$$

For models for which there is a fully deterministic path from $\mathbf{u}_t$ to $\mathbf{x}_t$ powerful enough to explain most of the structure in the data, such as the RNN in the VRNN and in the SRNN, it is possible that the model learns not use stochastic part at all. This was also noticed in (Chen et al., 2017) in the static setting for models with very expressive decoders. This issue is often accompanied in

the temporal setting both by small KL terms and very small learned variances of the distributions over the latent variables that make the model essentially deterministic. A possible way to partially prevent this from happening is by:

- Lower bounding the variances to a given value or fixing them, so that it is not possible for the model to completely disregard the stochasticity.
- Using some of the tricks presented above to avoid small KL terms.

A particular attention should be used for models in which the various components of the ELBO depend on vectors with very different dimensionalities. For KVAEs and GTMs-SM for example, the KL term depends on the low-dimensional SSM states whereas the observations in the VAE reconstruction term are high-dimensional images. In the ELBO therefore the reconstruction term will dominate, and the training algorithm will then mostly focus in optimizing the VAE and not the SSM. Possible solutions in this case are:

- Re-weighting of the terms in the ELBO, and in particular only considering a fraction of the reconstruction term of the VAEs during training (e.g. by multiplying it by 0.3). In this way we can in fact help the model to focus on learning the SSM temporal dynamics as well (Fraccaro et al., 2017).
- Alternate the updates of VAE and SSM parameters, so that we make sure that in some updates the SSM is being optimized as well (Fraccaro et al., 2017).

## 4.9 Summary and discussion

In this chapter we have introduced a family of sequential deep latent variable models for unsupervised learning of complex data distributions from large unlabelled datasets. In Sections 4.2 and 4.3 we have discussed recurrent neural networks and deep state-space models that, together with VAEs, form the building blocks for the more complex architectures presented in the rest of the chapter. The VAE-RNN model, introduced in Section 4.4.1, is a basic temporal extension of VAEs that uses an RNN to model the dependencies between latent variables at different time steps. In Section 4.4.2 we then argued that in some cases the VAE-RNN model is not enough to model accurately the temporal variability in the data, and defined the VRNN as a more expressive architecture that makes $\mathbf{z}_t$ depend on $\mathbf{z}_{t-1}$ indirectly through the deterministic RNN state $\mathbf{d}_t$. The SRNN model of Section 4.5 improves on the VRNN by adding a direct dependency between latent variables at consecutive time steps. The resulting architecture is then obtained stacking an RNN and a DSSM, and has a clear separation between deterministic and stochastic variables that is beneficial at inference time. We have then seen in Section 4.6 that using structured priors we can define architectures capable of learning disentangled representations that are more interpretable and data-efficient. In particular, we have constructed the KVAE model by using a LGSSM to parameterize the prior of the same VAE repeated at each time step, and showed that we can leverage the exact inference and missing data imputation capabilities of the LGSSM. To deal with applications that require a high memory capacity we have shown in Section 4.7 that we can exploit external memory architectures, such as the DND memory used in the GTM-SM

model. Due to the complexity of these models defining and training them can be challenging. In Section 4.8 we have then discussed some suggestions and training tricks that have proven useful when working with such models.

All the models presented in this chapter were obtained following the same very general procedure. We always start defining the joint distribution of the model in which we encode all the modelling assumptions that are suitable for the particular application at hand. To learn the parameters of the model using Maximum Likelihood, we need to compute the data log-likelihood by marginalizing the joint distribution over the latent variables. Since this integral is often intractable, we define a variational approximation over the latent variables of the model conditioned on the inputs and outputs, and use Jensen's inequality to derive the ELBO, the objective function used during training. When can design a variational distribution that better approximates the true posterior distribution by making use of the independence properties among the variables of the model. The scalability of the models is ensured using inference networks to define scalable and flexible variational approximations parameterized by deep neural networks. As both the generative model and the variational approximation are defined using neural network architectures, we can train their parameters jointly using stochastic gradient ascent, computing their gradients efficiently on GPU.

In the following chapters we will present in detail some of the models briefly described above, namely Stochastic recurrent neural networks (Fraccaro et al., 2016c, Chapter 5), Kalman variational auto-encoders (Fraccaro et al., 2017, Chapter 6) and generative temporal models with spatial memory (Fraccaro et al., 2018, Chapter 7).

# Part II

# Research papers

Chapter 5

# Sequential Neural Models with Stochastic Layers

Marco Fraccaro[a] · Søren Sønderby[b] · Ulrich Paquet[c] · Ole Winther[a,b]

[a]DTU Compute, Technical University of Denmark, Denmark

[b]Bioinformatics Centre, Department of Biology, University of Copenhagen, Denmark

[c]Google DeepMind, United Kingdom

**Publication Status:** Published in Advances in Neural Information Processing Systems 29, NIPS 2016. Selected for oral presentation.

**Abstract:** How can we efficiently propagate uncertainty in a latent state representation with recurrent neural networks? This paper introduces *stochastic recurrent neural networks* which glue a deterministic recurrent neural network and a state space model together to form a stochastic and sequential neural generative model. The clear separation of deterministic and stochastic layers allows a structured variational inference network to track the factorization of the model's posterior distribution. By retaining both the nonlinear recursive structure of a recurrent neural network and averaging over the uncertainty in a latent path, like a state space model, we improve the state of the art results on the Blizzard and TIMIT speech modeling data sets by a large margin, while achieving comparable performances to competing methods on polyphonic music modeling.

## 5.1 Introduction

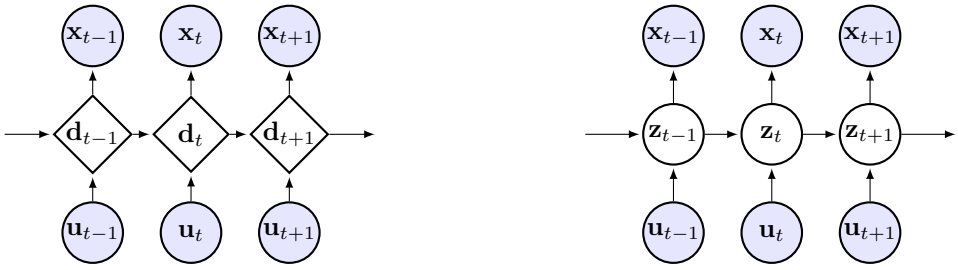
Recurrent neural networks (RNNs) are able to represent long-term dependencies in sequential data, by adapting and propagating a deterministic hidden (or latent) state (Cho et al., 2014; Hochreiter and Schmidhuber, 1997). There is recent evidence that when complex sequences such as speech and music are modeled, the performances of RNNs can be dramatically improved when uncertainty is included in their hidden states (Bayer and Osendorfer, 2014; Boulanger-Lewandowski et al., 2012; Chung et al., 2015; Fabius and Amersfoort, 2014; Gan et al., 2015; Gu et al., 2015). In this paper we add a new direction to the explorer's map of treating the hidden RNN states as uncertain paths, by including the world of state space models (SSMs) as an RNN layer. By cleanly delineating a SSM layer, certain independence properties of variables arise, which are beneficial for making efficient posterior inferences. The result is a generative model for sequential data, with a matching inference network that has its roots in variational auto-encoders (VAEs).

SSMs can be viewed as a probabilistic extension of RNNs, where the hidden states are assumed to be random variables. Although SSMs have an illustrious history (Roweis and Ghahramani, 1999), their stochasticity has limited their widespread use in the deep learning community, as inference can only be exact for two relatively simple classes of SSMs, namely hidden Markov models and linear Gaussian models, neither of which are well-suited to modeling long-term dependencies and complex probability distributions over high-dimensional sequences. On the other hand, modern RNNs rely on gated nonlinearities such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) cells or gated recurrent units (GRUs) (Chung et al., 2014), that let the deterministic hidden state of the RNN act as an internal memory for the model. This internal memory seems fundamental to capturing complex relationships in the data through a statistical model.

This paper introduces the *stochastic recurrent neural network (SRNN)* in Section 5.3. SRNNs combine the gated activation mechanism of RNNs with the stochastic states of SSMs, and are formed by stacking a RNN and a nonlinear SSM. The state transitions of the SSM are nonlinear and are parameterized by a neural network that also depends on the corresponding RNN hidden state. The SSM can therefore utilize long-term information captured by the RNN.

We use recent advances in variational inference to efficiently approximate the intractable posterior distribution over the latent states with an inference network (Kingma and Welling, 2014; Rezende et al., 2014). The form of our variational approximation is inspired by the independence properties of the true posterior distribution over the latent states of the model, and allows us to improve inference by conveniently using the information coming from the whole sequence at each time step. The posterior distribution over the latent states of the SRNN is highly non-stationary while we are learning the parameters of the model. To further improve the variational approximation, we show that we can construct the inference network so that it only needs to learn how to compute the mean of the variational approximation at each time step given the mean of the *predictive* prior distribution.

In Section 5.4 we test the performances of SRNN on speech and polyphonic music modeling tasks. SRNN improves the state of the art results on the Blizzard and TIMIT speech data sets by a large margin, and performs comparably to competing models on polyphonic music modeling. Finally, other models that extend RNNs by adding stochastic units will be reviewed and compared to




*(a) RNN* *(b) SSM*

***Figure 5.1:*** *Graphical models to generate $\mathbf{x}_{1:T}$ with a recurrent neural network (RNN) and a state space model (SSM). Diamond-shaped units are used for deterministic states, while circles are used for stochastic ones. For sequence generation, like in a language model, one can use $\mathbf{u}_t = \mathbf{x}_{t-1}$.*

SRNN in Section 5.5.

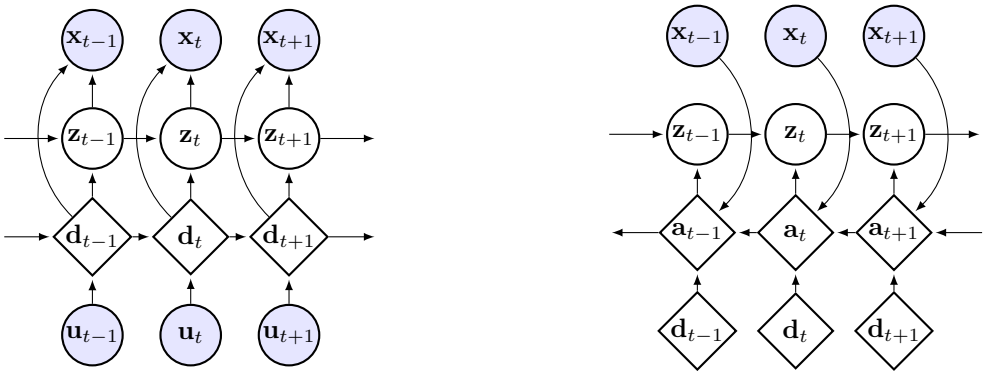
## 5.2 Recurrent Neural Networks and State Space Models

*Recurrent neural networks* and *state space models* are widely used to model temporal sequences of vectors $\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ that possibly depend on inputs $\mathbf{u}_{1:T} = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_T)$. Both models rest on the assumption that the sequence $\mathbf{x}_{1:t}$ of observations up to time $t$ can be summarized by a latent state $\mathbf{d}_t$ or $\mathbf{z}_t$, which is deterministically determined ($\mathbf{d}_t$ in a RNN) or treated as a random variable which is averaged away ($\mathbf{z}_t$ in a SSM). The difference in treatment of the latent state has traditionally led to vastly different models: RNNs recursively compute $\mathbf{d}_t = f(\mathbf{d}_{t-1}, \mathbf{u}_t)$ using a parameterized nonlinear function $f$, like a LSTM cell or a GRU. The RNN observation probabilities $p(\mathbf{x}_t|\mathbf{d}_t)$ are equally modeled with nonlinear functions. SSMs, like linear Gaussian or hidden Markov models, explicitly model uncertainty in the latent process through $\mathbf{z}_{1:T}$. Parameter inference in a SSM requires $\mathbf{z}_{1:T}$ to be averaged out, and hence $p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)$ and $p(\mathbf{x}_t|\mathbf{z}_t)$ are often restricted to the exponential family of distributions to make many existing approximate inference algorithms applicable. On the other hand, averaging a function over the deterministic path $\mathbf{d}_{1:T}$ in a RNN is a trivial operation. The striking similarity in factorization between these models is illustrated in Figures 5.1a and 5.1b.

Can we combine the best of both worlds, and make the stochastic state transitions of SSMs nonlinear whilst keeping the gated activation mechanism of RNNs? Below, we show that a more expressive model can be created by stacking a SSM on top of a RNN, and that by keeping them layered, the functional form of the true posterior distribution over $\mathbf{z}_{1:T}$ guides the design of a backward-recursive structured variational approximation.

## 5.3 Stochastic Recurrent Neural Networks

We define a SRNN as a generative model $p_\theta$ by temporally interlocking a SSM with a RNN, as illustrated in Figure 5.2a. The joint probability of a single sequence and its latent states,




***(a)*** *Generative model $p_\theta$* ***(b)*** *Inference network $q_\phi$*

***Figure 5.2:*** *A SRNN as a generative model $p_\theta$ for a sequence $\mathbf{x}_{1:T}$. Posterior inference of $\mathbf{z}_{1:T}$ and $\mathbf{d}_{1:T}$ is done through an inference network $q_\phi$, which uses a backward-recurrent state $\mathbf{a}_t$ to approximate the nonlinear dependence of $\mathbf{z}_t$ on future observations $\mathbf{x}_{t:T}$ and states $\mathbf{d}_{t:T}$; see Equation (5.7).*

assuming knowledge of the starting states $\mathbf{z}_0 = \mathbf{0}$ and $\mathbf{d}_0 = \mathbf{0}$, and inputs $\mathbf{u}_{1:T}$, factorizes as

$$\begin{aligned} p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T} | \mathbf{u}_{1:T}, \mathbf{z}_0, \mathbf{d}_0) &= p_{\theta_\mathrm{x}}(\mathbf{x}_{1:T} | \mathbf{z}_{1:T}, \mathbf{d}_{1:T})\, p_{\theta_\mathrm{z}}(\mathbf{z}_{1:T} | \mathbf{d}_{1:T}, \mathbf{z}_0)\, p_{\theta_\mathrm{d}}(\mathbf{d}_{1:T} | \mathbf{u}_{1:T}, \mathbf{d}_0) \\ &= \prod_{t=1}^{T} p_{\theta_\mathrm{x}}(\mathbf{x}_t | \mathbf{z}_t, \mathbf{d}_t)\, p_{\theta_\mathrm{z}}(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{d}_t)\, p_{\theta_\mathrm{d}}(\mathbf{d}_t | \mathbf{d}_{t-1}, \mathbf{u}_t) \ . \end{aligned} \tag{5.1}$$

The SSM and RNN are further tied with skip-connections from $\mathbf{d}_t$ to $\mathbf{x}_t$. The joint density in (5.1) is parameterized by $\theta = \{\theta_\mathrm{x}, \theta_\mathrm{z}, \theta_\mathrm{d}\}$, which will be adapted together with parameters $\phi$ of a so-called "inference network" $q_\phi$ to best model $N$ independently observed data sequences $\{\mathbf{x}^i_{1:T_i}\}_{i=1}^N$ that are described by the log marginal likelihood or evidence

$$\mathcal{L}(\theta) = \log p_\theta\left(\{\mathbf{x}^i_{1:T_i}\} \,|\, \{\mathbf{u}^i_{1:T_i}, \mathbf{z}^i_0, \mathbf{d}^i_0\}_{i=1}^N\right) = \sum_i \log p_\theta(\mathbf{x}^i_{1:T_i} | \mathbf{u}^i_{1:T_i}, \mathbf{z}^i_0, \mathbf{d}^i_0) = \sum_i \mathcal{L}_i(\theta) \ . \tag{5.2}$$

Throughout the paper, we omit superscript $i$ when only one sequence is referred to, or when it is clear from the context. In each log likelihood term $\mathcal{L}_i(\theta)$ in (5.2), the latent states $\mathbf{z}_{1:T}$ and $\mathbf{d}_{1:T}$ were averaged out of (5.1). Integrating out $\mathbf{d}_{1:T}$ is done by simply substituting its deterministically obtained value, but $\mathbf{z}_{1:T}$ requires more care, and we return to it in Section 5.3.2. Following Figure 5.2a, the states $\mathbf{d}_{1:T}$ are determined from $\mathbf{d}_0$ and $\mathbf{u}_{1:T}$ through the recursion $\mathbf{d}_t = f_{\theta_\mathrm{d}}(\mathbf{d}_{t-1}, \mathbf{u}_t)$. In our implementation $f_{\theta_\mathrm{d}}$ is a GRU network with parameters $\theta_\mathrm{d}$. For later convenience we denote the value of $\mathbf{d}_{1:T}$, as computed by application of $f_{\theta_\mathrm{d}}$, by $\widetilde{\mathbf{d}}_{1:T}$. Therefore $p_{\theta_\mathrm{d}}(\mathbf{d}_t | \mathbf{d}_{t-1}, \mathbf{u}_t) = \delta(\mathbf{d}_t - \widetilde{\mathbf{d}}_t)$, i.e. $\mathbf{d}_{1:T}$ follows a delta distribution centered at $\widetilde{\mathbf{d}}_{1:T}$.

Unlike the VRNN (Chung et al., 2015), $\mathbf{z}_t$ directly depends on $\mathbf{z}_{t-1}$, as it does in a SSM, via $p_{\theta_\mathrm{z}}(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{d}_t)$. This split makes a clear separation between the deterministic and stochastic parts of $p_\theta$; the RNN remains entirely deterministic and its recurrent units do not depend on noisy samples of $\mathbf{z}_t$, while the prior over $\mathbf{z}_t$ follows the Markov structure of SSMs. The split allows us to later mimic the structure of the posterior distribution over $\mathbf{z}_{1:T}$ and $\mathbf{d}_{1:T}$ in its approximation $q_\phi$. We let the prior transition distribution $p_{\theta_\mathrm{z}}(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{d}_t) = \mathcal{N}(\mathbf{z}_t; \boldsymbol{\mu}^{(p)}_t, \mathbf{v}^{(p)}_t)$ be a Gaussian with

a diagonal covariance matrix, whose mean and log-variance are parameterized by neural networks that depend on $\mathbf{z}_{t-1}$ and $\mathbf{d}_t$,

$$\boldsymbol{\mu}_t^{(p)} = \mathrm{NN}_1^{(p)}(\mathbf{z}_{t-1}, \mathbf{d}_t) \ , \qquad \log \mathbf{v}_t^{(p)} = \mathrm{NN}_2^{(p)}(\mathbf{z}_{t-1}, \mathbf{d}_t) \ , \tag{5.3}$$

where NN denotes a neural network. Parameters $\theta_\mathrm{z}$ denote all weights of $\mathrm{NN}_1^{(p)}$ and $\mathrm{NN}_2^{(p)}$, which are two-layer feed-forward networks in our implementation. Similarly, the parameters of the emission distribution $p_{\theta_\mathrm{x}}(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t)$ depend on $\mathbf{z}_t$ and $\mathbf{d}_t$ through a similar neural network that is parameterized by $\theta_\mathrm{x}$.

### 5.3.1 Variational inference for the SRNN

The stochastic variables $\mathbf{z}_{1:T}$ of the nonlinear SSM cannot be analytically integrated out to obtain $\mathcal{L}(\theta)$ in (5.2). Instead of maximizing $\mathcal{L}$ with respect to $\theta$, we maximize a variational *evidence lower bound* (ELBO) $\mathcal{F}(\theta, \phi) = \sum_i \mathcal{F}_i(\theta, \phi) \leq \mathcal{L}(\theta)$ with respect to both $\theta$ and the variational parameters $\phi$ (Jordan et al., 1999). The ELBO is a sum of lower bounds $\mathcal{F}_i(\theta, \phi) \leq \mathcal{L}_i(\theta)$, one for each sequence $i$,

$$\mathcal{F}_i(\theta, \phi) = \iint q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, A) \log \frac{p_\theta(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{d}_{1:T}|A)}{q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, A)} \, \mathrm{d}\mathbf{z}_{1:T} \, \mathrm{d}\mathbf{d}_{1:T} \ , \tag{5.4}$$

where $A = \{\mathbf{u}_{1:T}, \mathbf{z}_0, \mathbf{d}_0\}$ is a notational shorthand. Each sequence's approximation $q_\phi$ shares parameters $\phi$ with all others, to form the *auto-encoding variational Bayes* inference network or variational auto encoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014) shown in Figure 5.2b. Maximizing $\mathcal{F}(\theta, \phi)$ – which we call "training" the neural network architecture with parameters $\theta$ and $\phi$ – is done by stochastic gradient ascent, and in doing so, both the posterior and its approximation $q_\phi$ change simultaneously. All the intractable expectations in (5.4) would typically be approximated by sampling, using the reparameterization trick (Kingma and Welling, 2014; Rezende et al., 2014) or control variates (Paisley et al., 2012) to obtain low-variance estimators of its gradients. We use the reparameterization trick in our implementation. Iteratively maximizing $\mathcal{F}$ over $\theta$ and $\phi$ separately would yield an expectation maximization-type algorithm, which has formed a backbone of statistical modeling for many decades (Dempster et al., 1977). The tightness of the bound depends on how well we can approximate the $i = 1, \ldots, N$ factors $p_\theta(\mathbf{z}^i_{1:T_i}, \mathbf{d}^i_{1:T_i}|\mathbf{x}^i_{1:T_i}, A^i)$ that constitute the true posterior over all latent variables with their corresponding factors $q_\phi(\mathbf{z}^i_{1:T_i}, \mathbf{d}^i_{1:T_i}|\mathbf{x}^i_{1:T_i}, A^i)$. In what follows, we show how $q_\phi$ could be judiciously structured to match the posterior factors.

We add initial structure to $q_\phi$ by noticing that the prior $p_{\theta_\mathrm{d}}(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0)$ in the generative model is a delta function over $\widetilde{\mathbf{d}}_{1:T}$, and so is the posterior $p_\theta(\mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)$. Consequently, we let the inference network use exactly the same deterministic state setting $\widetilde{\mathbf{d}}_{1:T}$ as that of the generative model, and we decompose it as

$$q_\phi(\mathbf{z}_{1:T}, \mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0, \mathbf{d}_0) = q_\phi(\mathbf{z}_{1:T}|\mathbf{d}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0) \underbrace{q(\mathbf{d}_{1:T}|\mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{d}_0)}_{= p_{\theta_\mathrm{d}}(\mathbf{d}_{1:T}|\mathbf{u}_{1:T}, \mathbf{d}_0)} \ . \tag{5.5}$$

This choice exactly approximates one delta-function by itself, and simplifies the ELBO by letting them cancel out. By further taking the outer average in (5.4), one obtains

$$\mathcal{F}_i(\theta, \phi) = \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}, \widetilde{\mathbf{d}}_{1:T})\right] - \mathrm{KL}\left(q_\phi(\mathbf{z}_{1:T}|\widetilde{\mathbf{d}}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0) \,\middle\|\, p_\theta(\mathbf{z}_{1:T}|\widetilde{\mathbf{d}}_{1:T}, \mathbf{z}_0)\right) \ , \tag{5.6}$$

which still depends on $\theta_\mathrm{d}$, $\mathbf{u}_{1:T}$ and $\mathbf{d}_0$ via $\widetilde{\mathbf{d}}_{1:T}$. The first term is an expected log likelihood under $q_\phi(\mathbf{z}_{1:T}|\widetilde{\mathbf{d}}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0)$, while KL denotes the Kullback-Leibler divergence between two distributions. Having stated the second factor in (5.5), we now turn our attention to parameterizing the first factor in (5.5) to resemble its posterior equivalent, by exploiting the temporal structure of $p_\theta$.

### 5.3.2 Exploiting the temporal structure

The true posterior distribution of the stochastic states $\mathbf{z}_{1:T}$, given both the data *and* the deterministic states $\mathbf{d}_{1:T}$, factorizes as $p_\theta(\mathbf{z}_{1:T}|\mathbf{d}_{1:T}, \mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_0) = \prod_t p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t:T}, \mathbf{x}_{t:T})$. This can be verified by considering the conditional independence properties of the graphical model in Figure 5.2a using *d-separation* (Geiger et al., 1990). This shows that, knowing $\mathbf{z}_{t-1}$, the posterior distribution of $\mathbf{z}_t$ does not depend on the past outputs and deterministic states, but only on the present and future ones; this was also noted in (Krishnan et al., 2015). Instead of factorizing $q_\phi$ as a mean-field approximation across time steps, we keep the structured form of the posterior factors, including $\mathbf{z}_t$'s dependence on $\mathbf{z}_{t-1}$, in the variational approximation

$$\begin{aligned} q_\phi(\mathbf{z}_{1:T}|\mathbf{d}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0) &= \prod_t q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t:T}, \mathbf{x}_{t:T}) \\ &= \prod_t q_{\phi_\mathrm{z}}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t = g_{\phi_\mathrm{a}}(\mathbf{a}_{t+1}, [\mathbf{d}_t, \mathbf{x}_t])) \;, \end{aligned} \tag{5.7}$$

where $[\mathbf{d}_t, \mathbf{x}_t]$ is the concatenation of the vectors $\mathbf{d}_t$ and $\mathbf{x}_t$. The graphical model for the inference network is shown in Figure 5.2b. Apart from the direct dependence of the posterior approximation at time $t$ on both $\mathbf{d}_{t:T}$ and $\mathbf{x}_{t:T}$, the distribution also depends on $\mathbf{d}_{1:t-1}$ and $\mathbf{x}_{1:t-1}$ through $\mathbf{z}_{t-1}$. We mimic each posterior factor's nonlinear long-term dependence on $\mathbf{d}_{t:T}$ and $\mathbf{x}_{t:T}$ through a backward-recurrent function $g_{\phi_\mathrm{a}}$, shown in (5.7), which we will return to in greater detail in Section 5.3.3. The inference network in Figure 5.2b is therefore parameterized by $\phi = \{\phi_\mathrm{z}, \phi_\mathrm{a}\}$ and $\theta_\mathrm{d}$.

In (5.7) all time steps are taken into account when constructing the variational approximation at time $t$; this can therefore be seen as a *smoothing* problem. In our experiments we also consider *filtering*, where only the information up to time $t$ is used to define $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{d}_t, \mathbf{x}_t)$. As the parameters $\phi$ are shared across time steps, we can easily handle sequences of variable length in both cases.

As both the generative model and inference network factorize over time steps in (5.1) and (5.7), the ELBO in (5.6) separates as a sum over the time steps

$$\begin{aligned} \mathcal{F}_i(\theta, \phi) = \sum_t \mathbb{E}_{q^*_\phi(\mathbf{z}_{t-1})} \Big[ \mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \widetilde{\mathbf{d}}_{t:T}, \mathbf{x}_{t:T})} \big[ \log p_\theta(\mathbf{x}_t|\mathbf{z}_t, \widetilde{\mathbf{d}}_t) \big] + \\ - \mathrm{KL}\Big( q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \widetilde{\mathbf{d}}_{t:T}, \mathbf{x}_{t:T}) \,\Big\|\, p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \widetilde{\mathbf{d}}_t) \Big) \Big] \;, \end{aligned} \tag{5.8}$$

where $q^*_\phi(\mathbf{z}_{t-1})$ denotes the marginal distribution of $\mathbf{z}_{t-1}$ in the variational approximation to the posterior $q_\phi(\mathbf{z}_{1:t-1}|\widetilde{\mathbf{d}}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0)$, given by

$$\begin{aligned} q^*_\phi(\mathbf{z}_{t-1}) &= \int q_\phi(\mathbf{z}_{1:t-1}|\widetilde{\mathbf{d}}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0) \, \mathrm{d}\mathbf{z}_{1:t-2} \\ &= \mathbb{E}_{q^*_\phi(\mathbf{z}_{t-2})} \Big[ q_\phi(\mathbf{z}_{t-1}|\mathbf{z}_{t-2}, \widetilde{\mathbf{d}}_{t-1:T}, \mathbf{x}_{t-1:T}) \Big] \;. \end{aligned} \tag{5.9}$$

We can interpret (5.9) as having a VAE at each time step $t$, with the VAE being conditioned on the past through the stochastic variable $\mathbf{z}_{t-1}$. To compute (5.8), the dependence on $\mathbf{z}_{t-1}$ needs to be integrated out, using our posterior knowledge at time $t-1$ which is given by $q_\phi^*(\mathbf{z}_{t-1})$. We approximate the outer expectation in (5.8) using a Monte Carlo estimate, as samples from $q_\phi^*(\mathbf{z}_{t-1})$ can be efficiently obtained by ancestral sampling. The sequential formulation of the inference model in (5.7) allows such samples to be drawn and reused, as given a sample $\mathbf{z}_{t-2}^{(s)}$ from $q_\phi^*(\mathbf{z}_{t-2})$, a sample $\mathbf{z}_{t-1}^{(s)}$ from $q_\phi(\mathbf{z}_{t-1}|\mathbf{z}_{t-2}^{(s)}, \widetilde{\mathbf{d}}_{t-1:T}, \mathbf{x}_{t-1:T})$ will be distributed according to $q_\phi^*(\mathbf{z}_{t-1})$.

### 5.3.3 Parameterization of the inference network

The variational distribution $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t:T}, \mathbf{x}_{t:T})$ needs to approximate the dependence of the true posterior $p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{d}_{t:T}, \mathbf{x}_{t:T})$ on $\mathbf{d}_{t:T}$ and $\mathbf{x}_{t:T}$, and as alluded to in (5.7), this is done by running a RNN with inputs $\mathbf{d}_{t:T}$ and $\mathbf{x}_{t:T}$ backwards in time. Specifically, we initialize the hidden state of the backward-recursive RNN in Figure 5.2b as $\mathbf{a}_{T+1} = \mathbf{0}$, and recursively compute $\mathbf{a}_t = g_{\phi_\mathrm{a}}(\mathbf{a}_{t+1}, [\widetilde{\mathbf{d}}_t, \mathbf{x}_t])$. The function $g_{\phi_\mathrm{a}}$ represents a recurrent neural network with, for example, LSTM or GRU units. Each sequence's variational approximation factorizes over time with $q_\phi(\mathbf{z}_{1:T}|\mathbf{d}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0) = \prod_t q_{\phi_\mathrm{z}}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t)$, as shown in (5.7). We let $q_{\phi_\mathrm{z}}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t)$ be a Gaussian with diagonal covariance, whose mean and the log-variance are parameterized with $\phi_\mathrm{z}$ as

$$\boldsymbol{\mu}_t^{(q)} = \mathrm{NN}_1^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t) \;, \qquad \log \mathbf{v}_t^{(q)} = \mathrm{NN}_2^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t) \;. \tag{5.10}$$

Instead of smoothing, we can also do filtering by using a neural network to approximate the dependence of the true posterior $p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{d}_t, \mathbf{x}_t)$ on $\mathbf{d}_t$ and $\mathbf{x}_t$, through for instance $\mathbf{a}_t = \mathrm{NN}^{(a)}(\mathbf{d}_t, \mathbf{x}_t)$.

**Improving the posterior approximation.** In our experiments we found that during training, the parameterization introduced in (5.10) can lead to small values of the KL term $\mathrm{KL}(q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t) \,\|\, p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \widetilde{\mathbf{d}}_t))$ in the ELBO in (5.8). This happens when $g_\phi$ in the inference network does not rely on the information propagated back from future outputs in $\mathbf{a}_t$, but it is mostly using the hidden state $\widetilde{\mathbf{d}}_t$ to imitate the behavior of the prior. The inference network could therefore get stuck by trying to optimize the ELBO through sampling from the prior of the model, making the variational approximation to the posterior useless. To overcome this issue, we directly include some knowledge of the *predictive* prior dynamics in the parameterization of the inference network, using our approximation of the posterior distribution $q_\phi^*(\mathbf{z}_{t-1})$ over the previous latent states. In the spirit of *sequential Monte Carlo* methods (Doucet et al., 2001), we improve the parameterization of $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t)$ by using $q_\phi^*(\mathbf{z}_{t-1})$ from (5.9). As we are constructing the variational distribution sequentially, we approximate the predictive prior mean, i.e. our "best guess" on the prior dynamics of $\mathbf{z}_t$, as

$$\widehat{\boldsymbol{\mu}}_t^{(p)} = \int \mathrm{NN}_1^{(p)}(\mathbf{z}_{t-1}, \mathbf{d}_t)\, p(\mathbf{z}_{t-1}|\mathbf{x}_{1:T})\, \mathrm{d}\mathbf{z}_{t-1} \approx \int \mathrm{NN}_1^{(p)}(\mathbf{z}_{t-1}, \mathbf{d}_t)\, q_\phi^*(\mathbf{z}_{t-1})\, \mathrm{d}\mathbf{z}_{t-1} \;, \tag{5.11}$$

where we used the parameterization of the prior distribution in (5.3). We estimate the integral required to compute $\widehat{\boldsymbol{\mu}}_t^{(p)}$ by reusing the samples that were needed for the Monte Carlo estimate

**Algorithm 2** Inference of SRNN with $\text{Res}_q$ parameterization from (5.12).

1: **inputs**: $\widetilde{\mathbf{d}}_{1:T}$ and $\mathbf{a}_{1:T}$
2: initialize $\mathbf{z}_0$
3: **for** $t = 1$ to $T$ **do**
4: $\quad \widehat{\boldsymbol{\mu}}_t^{(p)} = \text{NN}_1^{(p)}(\mathbf{z}_{t-1}, \widetilde{\mathbf{d}}_t)$
5: $\quad \boldsymbol{\mu}_t^{(q)} = \widehat{\boldsymbol{\mu}}_t^{(p)} + \text{NN}_1^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t)$
6: $\quad \log \mathbf{v}_t^{(q)} = \text{NN}_2^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t)$
7: $\quad \mathbf{z}_t \sim \mathcal{N}(\mathbf{z}_t; \boldsymbol{\mu}_t^{(q)}, \mathbf{v}_t^{(q)})$
8: **end for**

of the ELBO in (5.8). This predictive prior mean can then be used in the parameterization of the mean of the variational approximation $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t)$,

$$\boldsymbol{\mu}_t^{(q)} = \widehat{\boldsymbol{\mu}}_t^{(p)} + \text{NN}_1^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t) \ , \tag{5.12}$$

and we refer to this parameterization as $\text{Res}_q$ in the results in Section 5.4. Rather than directly learning $\boldsymbol{\mu}_t^{(q)}$, we learn the *residual* between $\widehat{\boldsymbol{\mu}}_t^{(p)}$ and $\boldsymbol{\mu}_t^{(q)}$. It is straightforward to show that with this parameterization the KL-term in (5.8) will not depend on $\widehat{\boldsymbol{\mu}}_t^{(p)}$, but only on $\text{NN}_1^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t)$. Learning the residual improves inference, making it seemingly easier for the inference network to track changes in the generative model while the model is trained, as it will only have to learn how to "correct" the predictive prior dynamics by using the information coming from $\widetilde{\mathbf{d}}_{t:T}$ and $\mathbf{x}_{t:T}$. We did not see any improvement in results by parameterizing $\log \mathbf{v}_t^{(q)}$ in a similar way. The inference procedure of SRNN with $\text{Res}_q$ parameterization for one sequence is summarized in Algorithm 2.

## 5.4 Results

In this section the SRNN is evaluated on the modeling of speech and polyphonic music data, as they have shown to be difficult to model without a good representation of the uncertainty in the latent states Bayer and Osendorfer, 2014; Chung et al., 2015; Fabius and Amersfoort, 2014; Gan et al., 2015; Gu et al., 2015. We test SRNN on the Blizzard (King and Karaiskos, 2013) and TIMIT raw audio data sets (Table 5.1) used in (Chung et al., 2015). The preprocessing of the data sets and the testing performance measures are identical to those reported in (Chung et al., 2015). Blizzard is a dataset of 300 hours of English, spoken by a single female speaker. TIMIT is a dataset of 6300 English sentences read by 630 speakers. As done in (Chung et al., 2015), for Blizzard we report the average log-likelihood for half-second sequences and for TIMIT we report the average log likelihood per sequence for the test set sequences. Note that the sequences in the TIMIT test set are on average 3.1s long, and therefore 6 times longer than those in Blizzard. For the raw audio datasets we use a fully factorized Gaussian output distribution. Additionally, we test SRNN for modeling sequences of polyphonic music (Table 5.2), using the four data sets of MIDI songs introduced in (Boulanger-Lewandowski et al., 2012). Each data set contains more than 7 hours of polyphonic music of varying complexity: folk tunes (Nottingham data set), the four-part chorales by J. S. Bach (JSB chorales), orchestral music (MuseData) and classical piano music (Piano-midi.de). For polyphonic music we use a Bernoulli output distribution to model the

binary sequences of piano notes. In our experiments we set $\mathbf{u}_t = \mathbf{x}_{t-1}$, but $\mathbf{u}_t$ could also be used to represent additional input information to the model.

All models where implemented using Theano (Bastien et al., 2012), Lasagne (Dieleman et al., 2015) and Parmesan[1]. Training using a NVIDIA Titan X GPU took around 1.5 hours for TIMIT, 18 hours for Blizzard, less than 15 minutes for the JSB chorales and Piano-midi.de data sets, and around 30 minutes for the Nottingham and MuseData data sets. To reduce the computational requirements we use only 1 sample to approximate all the intractable expectations in the ELBO (notice that the KL term can be computed analytically). Further implementation and experimental details can be found in the Supplementary Material.

**Blizzard and TIMIT.** Table 5.1 compares the average log-likelihood per test sequence of SRNN to the results from (Chung et al., 2015). For RNNs and VRNNs the authors of (Chung et al., 2015) test two different output distributions, namely a Gaussian distribution (Gauss) and a Gaussian Mixture Model (GMM). VRNN-I differs from the VRNN in that the prior over the latent variables is independent across time steps, and it is therefore similar to STORN (Bayer and Osendorfer, 2014). For SRNN we compare the *smoothing* and *filtering* performance (denoted as *smooth* and *filt* in Table 5.1), both with the residual term from (5.12) and without it (5.10) (denoted as $\text{Res}_q$ if present). We prefer to only report the more conservative evidence lower bound for SRNN, as the approximation of the log-likelihood using standard importance sampling is known to be difficult to compute accurately in the sequential setting (Doucet et al., 2001). We see from Table 5.1 that SRNN outperforms all the competing methods for speech modeling. As the test sequences in TIMIT are on average more than 6 times longer than the ones for Blizzard, the results obtained with SRNN for TIMIT are in line with those obtained for Blizzard. The VRNN, which performs well when the voice of the single speaker from Blizzard is modeled, seems to encounter difficulties when modeling the 630 speakers in the TIMIT data set. As expected, for SRNN the variational approximation that is obtained when future information is also used (smoothing) is better than the one obtained by filtering. Learning the residual between the prior mean and the mean of the variational approximation, given in (5.12), further improves the performance in 3 out of 4 cases.

In the first two lines of Figure 5.3 we plot two raw signals from the Blizzard test set and the average KL term between the variational approximation and the prior distribution. We see that the KL term increases whenever there is a transition in the raw audio signal, meaning that the inference network is using the information coming from the output symbols to improve inference. Finally, the reconstructions of the output mean and log-variance in the last two lines of Figure 5.3 look consistent with the original signal.

**Polyphonic music.** Table 5.2 compares the average log-likelihood on the test sets obtained with SRNN and the models introduced in (Bayer and Osendorfer, 2014; Boulanger-Lewandowski et al., 2012; Gan et al., 2015; Gu et al., 2015). As done for the speech data, we prefer to report the more conservative estimate of the ELBO in Table 5.2, rather than approximating the log-likelihood with importance sampling as some of the other methods do. We see that SRNN performs comparably to other state of the art methods in all four data sets. We report the results using smoothing and learning the residual between the mean of the predictive prior and the

[1] github.com/casperkaae/parmesan. The code for SRNN is available at github.com/marcofraccaro/srnn.

| Models | Blizzard | TIMIT |
|---|---|---|
| SRNN (smooth+Res$_q$) | $\geq$**11991** | $\geq$ **60550** |
| SRNN (smooth) | $\geq$ 10991 | $\geq$ 59269 |
| SRNN (filt+Res$_q$) | $\geq$ 10572 | $\geq$ 52126 |
| SRNN (filt) | $\geq$ 10846 | $\geq$ 50524 |
| VRNN-GMM | $\geq$ 9107 | $\geq$ 28982 |
| | $\approx$ 9392 | $\approx$ 29604 |
| VRNN-Gauss | $\geq$ 9223 | $\geq$ 28805 |
| | $\approx$ 9516 | $\approx$ 30235 |
| VRNN-I-Gauss | $\geq$ 8933 | $\geq$ 28340 |
| | $\approx$ 9188 | $\approx$ 29639 |
| RNN-GMM | 7413 | 26643 |
| RNN-Gauss | 3539 | -1900 |

***Table 5.1:*** *Average log-likelihood per sequence on the test sets. For TIMIT the average test set length is 3.1s, while the Blizzard sequences are all 0.5s long. The non-SRNN results are reported as in (Chung et al., 2015). Smooth: $g_{\phi_a}$ is a GRU running backwards; filt: $g_{\phi_a}$ is a feed-forward network; Res$_q$: parameterization with residual in (5.12).*

| Models | Nottingham | JSB chorales | MuseData | Piano-midi.de |
|---|---|---|---|---|
| SRNN (smooth+Res$_q$) | $\geq -2.94$ | $\geq -4.74$ | $\geq -6.28$ | $\geq -8.20$ |
| TSBN | $\geq -3.67$ | $\geq -7.48$ | $\geq -6.81$ | $\geq -7.98$ |
| NASMC | $\approx -2.72$ | $\approx$ **$-3.99$** | $\approx -6.89$ | $\approx -7.61$ |
| STORN | $\approx -2.85$ | $\approx -6.91$ | $\approx -6.16$ | $\approx -7.13$ |
| RNN-NADE | $\approx$ **$-2.31$** | $\approx -5.19$ | $\approx$ **$-5.60$** | $\approx$ **$-7.05$** |
| RNN | $\approx -4.46$ | $\approx -8.71$ | $\approx -8.13$ | $\approx -8.37$ |

***Table 5.2:*** *Average log-likelihood on the test sets. The TSBN results are from (Gan et al., 2015), NASMC from (Gu et al., 2015), STORN from (Bayer and Osendorfer, 2014), RNN-NADE and RNN from (Boulanger-Lewandowski et al., 2012).*

mean of the variational approximation, but the performances using filtering and directly learning the mean of the variational approximation are now similar. We believe that this is due to the small amount of data and the fact that modeling MIDI music is much simpler than modeling raw speech signals.

## 5.5 Related work

A number of works have extended RNNs with stochastic units to model motion capture, speech and music data (Bayer and Osendorfer, 2014; Chung et al., 2015; Fabius and Amersfoort, 2014; Gan et al., 2015; Gu et al., 2015). The performances of these models are highly dependent on how the dependence among stochastic units is modeled over time, on the type of interaction between stochastic units and deterministic ones, and on the procedure that is used to evaluate the typically intractable log likelihood. Figure 5.4 highlights how SRNN differs from some of these works.

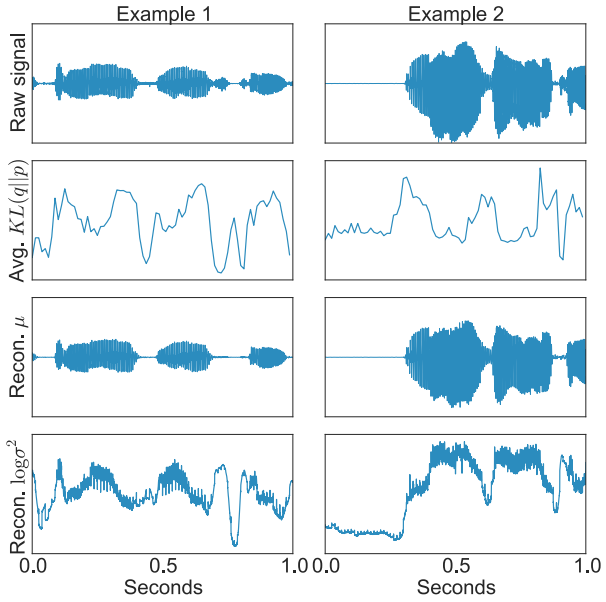
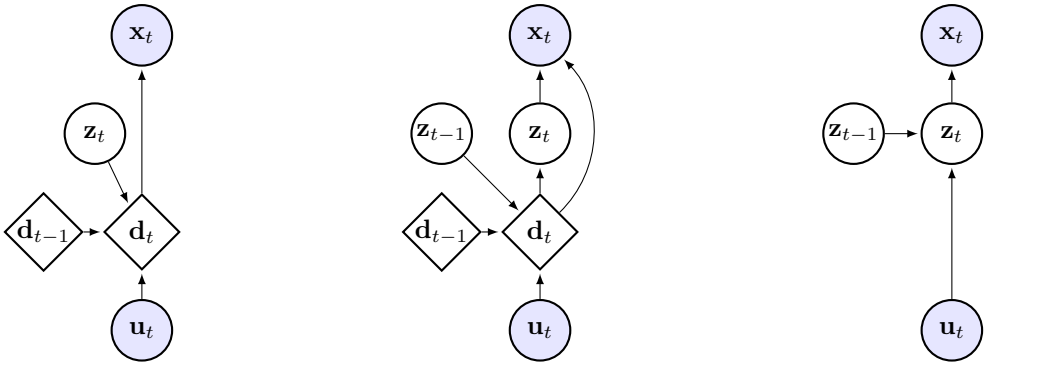



***Figure 5.3:*** *Visualization of the average KL term and reconstructions of the output mean and log-variance for two examples from the Blizzard test set.*




***(a)*** *STORN* ***(b)*** *VRNN* ***(c)*** *Deep Kalman Filter*

***Figure 5.4:*** *Generative models of $\mathbf{x}_{1:T}$ that are related to SRNN. For sequence modeling it is typical to set $\mathbf{u}_t = \mathbf{x}_{t-1}$.*

In STORN (Bayer and Osendorfer, 2014) *(Figure 5.4a)* and DRAW (Gregor et al., 2015) the stochastic units at each time step have an isotropic Gaussian prior and are independent between time steps. The stochastic units are used as an input to the deterministic units in a RNN. As in our work, the reparameterization trick (Kingma and Welling, 2014; Rezende et al., 2014) is used to optimize an ELBO.

The authors of the VRNN (Chung et al., 2015) *(Figure 5.4b)* note that it is beneficial to add information coming from the past states to the prior over latent variables $\mathbf{z}_t$. The VRNN lets the prior $p_{\theta_z}(\mathbf{z}_t|\mathbf{d}_t)$ over the stochastic units depend on the deterministic units $\mathbf{d}_t$, which in turn depend on both the deterministic and the stochastic units at the previous time step through the recursion $\mathbf{d}_t = f(\mathbf{d}_{t-1}, \mathbf{z}_{t-1}, \mathbf{u}_t)$. The SRNN differs by clearly separating the deterministic and stochastic part, as shown in Figure 5.2a. The separation of deterministic and stochastic units allows us to improve the posterior approximation by doing smoothing, as the stochastic units still

depend on each other when we condition on $\mathbf{d}_{1:T}$. In the VRNN, on the other hand, the stochastic units are conditionally independent given the states $\mathbf{d}_{1:T}$. Because the inference and generative networks in the VRNN share the deterministic units, the variational approximation would not improve by making it dependent on the future through $\mathbf{a}_t$, when calculated with a backward GRU, as we do in our model. Unlike STORN, DRAW and VRNN, the SRNN separates the "noisy" stochastic units from the deterministic ones, forming an entire layer of interconnected stochastic units. We found in practice that this gave better performance and was easier to train. The works by (Archer et al., 2015; Krishnan et al., 2015) *(Figure 5.4c)* show that it is possible to improve inference in SSMs by using ideas from VAEs, similar to what is done in the stochastic part (the top layer) of SRNN. Towards the periphery of related works, (Gu et al., 2015) approximates the log likelihood of a SSM with sequential Monte Carlo, by learning flexible proposal distributions parameterized by deep networks, while (Gan et al., 2015) uses a recurrent model with discrete stochastic units that is optimized using the NVIL algorithm (Mnih and Gregor, 2014).

## 5.6 Conclusion

This work has shown how to extend the modeling capabilities of recurrent neural networks by combining them with nonlinear state space models. Inspired by the independence properties of the intractable true posterior distribution over the latent states, we designed an inference network in a principled way. The variational approximation for the stochastic layer was improved by using the information coming from the whole sequence and by using the $\text{Res}_q$ parameterization to help the inference network to track the non-stationary posterior. SRNN achieves state of the art performances on the Blizzard and TIMIT speech data set, and performs comparably to competing methods for polyphonic music modeling.

## Supplementary material

### Experimental setup

**Blizzard and TIMIT.** The sampling rate is 16KHz and the raw audio signal is normalized using the global mean and standard deviation of the traning set. We split the raw audio signals in chunks of 2 seconds. The waveforms are then divided into non-overlapping vectors of size 200. The RNN thus runs for 160 steps[2]. The model is trained to predict the next vector ($\mathbf{x}_t$) given the current one ($\mathbf{u}_t$). During training we use backpropagation through time (BPTT) for 0.5 seconds, i.e we have 4 updates for each 2 seconds of audio. For the first 0.5 second we initialize hidden units with zeros and for the subsequent 3 chunks we use the previous hidden states as initialization.

For Blizzard we split the data using 90% for training, 5% for validation and 5% for testing. For testing we report the average log-likelihood per 0.5s sequences. For TIMIT we use the predefined test set for testing and split the rest of the data into 95% for training and 5% for validation. The

[2] 2s·16Khz / 200 = 160

training and testing setup are identical to the ones for Blizzard. For TIMIT the test sequences have variable length and are on average 3.1$s$, i.e. more than 6 times longer than Blizzard.

We model the output using a fully factorized Gaussian distribution for $p_{\theta_x}(\mathbf{x}_t|\mathbf{z}_t, \mathbf{d}_t)$. The deterministic RNNs use GRUs (Chung et al., 2014), with 2048 units for Blizzard and 1024 units for TIMIT. In both cases, $\mathbf{z}_t$ is a 256-dimensional vector. All the neural networks have 2 layers, with 1024 units for Blizzard and 512 for TIMIT, and use leaky rectified nonlinearities with leakiness $\frac{1}{3}$ and clipped at $\pm 3$. In both generative and inference models we share a neural network to extract features from the raw audio signal. The sizes of the models were chosen to roughly match the number of parameters used in (Chung et al., 2015). In all experiments it was fundamental to gradually introduce the KL term in the ELBO, as shown in (Bowman et al., 2015; Sønderby et al., 2016a; Raiko et al., 2007). We therefore multiply a temperature $\beta$ to the KL term, i.e. $\beta$KL, and linearly increase $\beta$ from 0.2 to 1 in the beginning of training (for Blizzard we increase it by 0.0001 after each update, while for TIMIT by 0.0003). In both data sets we used the ADAM optimizer (Kingma and Ba, 2014). For Blizzard we use a learning rate of 0.0003 and batch size of 128, for TIMIT they are 0.001 and 64 respectively.

**Polyphonic music.** We use the same model architecture as in Section 5.4, except for the output Bernoulli variables used to model the active notes. We reduced the number of parameters in the model to 300 deterministic hidden units for the GRU networks, and 100 stochastic units whose distributions are parameterized with neural networks with 1 layer of 500 units.

Chapter 6

# A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning

Marco Fraccaro[a] · Simon Kamronn[a] · Ulrich Paquet[b] · Ole Winther[a]

[a]DTU Compute, Technical University of Denmark, Denmark

[b]Google DeepMind, United Kingdom

**Publication Status:** Published in Advances in Neural Information Processing Systems 30, NIPS 2017. Selected for spotlight presentation.

**Abstract:** This paper takes a step towards temporal reasoning in a dynamically changing video, not in the pixel space that constitutes its frames, but in a latent space that describes the non-linear dynamics of the objects in its world. We introduce the Kalman variational auto-encoder, a framework for unsupervised learning of sequential data that disentangles two latent representations: an object's representation, coming from a recognition model, and a latent state describing its dynamics. As a result, the evolution of the world can be imagined and missing data imputed, both without the need to generate high dimensional frames at each time step. The model is trained end-to-end on videos of a variety of simulated physical systems, and outperforms competing methods in generative and missing data imputation tasks.

## 6.1 Introduction

From the earliest stages of childhood, humans learn to represent high-dimensional sensory input to make temporal predictions. From the visual image of a moving tennis ball, we can imagine its trajectory, and prepare ourselves in advance to catch it. Although the act of recognising the tennis ball is seemingly independent of our intuition of Newtonian dynamics (L. G. Ungerleider and L. G. Haxby, 1994), very little of this assumption has yet been captured in the end-to-end models that presently mark the path towards artificial general intelligence. Instead of basing inference on any abstract grasp of dynamics that is learned from experience, current successes are autoregressive: to imagine the tennis ball's trajectory, one forward-generates a frame-by-frame rendering of the full sensory input (Chiappa et al., 2017; Finn et al., 2016; Oh et al., 2015; Patraucean et al., 2015; Srivastava et al., 2015; Sun et al., 2016).

To disentangle two latent representations, an object's, and that of its dynamics, this paper introduces *Kalman variational auto-encoders (KVAEs)*, a model that separates an intuition of dynamics from an object recognition network (section 6.3). At each time step $t$, a variational auto-encoder (Kingma and Welling, 2014; Rezende et al., 2014) compresses high-dimensional visual stimuli $\mathbf{x}_t$ into latent encodings $\mathbf{a}_t$. The temporal dynamics in the learned $\mathbf{a}_t$-manifold are modelled with a linear Gaussian state space model that is adapted to handle complex dynamics (despite the linear relations among its states $\mathbf{z}_t$). The parameters of the state space model are adapted at each time step, and non-linearly depend on past $\mathbf{a}_t$'s via a recurrent neural network. Exact posterior inference for the linear Gaussian state space model can be performed with the Kalman filtering and smoothing algorithms, and is used for imputing missing data, for instance when we imagine the trajectory of a bouncing ball after observing it in initial and final video frames (section 6.4). The separation between recognition and dynamics model allows for missing data imputation to be done via a combination of the latent states $\mathbf{z}_t$ of the model and its encodings $\mathbf{a}_t$ only, without having to forward-sample high-dimensional images $\mathbf{x}_t$ in an autoregressive way. KVAEs are tested on videos of a variety of simulated physical systems in section 6.5: from raw visual stimuli, it "end-to-end" learns the interplay between the recognition and dynamics components. As KVAEs can do smoothing, they outperform an array of methods in generative and missing data imputation tasks (section 6.5).

## 6.2 Background

**Linear Gaussian state space models.** Linear Gaussian state space models (LGSSMs) are widely used to model sequences of vectors $\mathbf{a} = \mathbf{a}_{1:T} = [\mathbf{a}_1, .., \mathbf{a}_T]$. LGSSMs model temporal correlations through a first-order Markov process on latent states $\mathbf{z} = [\mathbf{z}_1, .., \mathbf{z}_T]$, which are potentially further controlled with external inputs $\mathbf{u} = [\mathbf{u}_1, .., \mathbf{u}_T]$, through the Gaussian distributions

$$p_{\gamma_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t; \mathbf{A}_t\mathbf{z}_{t-1} + \mathbf{B}_t\mathbf{u}_t, \mathbf{Q}) \tag{6.1}$$

$$p_{\gamma_t}(\mathbf{a}_t|\mathbf{z}_t) = \mathcal{N}(\mathbf{a}_t; \mathbf{C}_t\mathbf{z}_t, \mathbf{R}) \ . \tag{6.2}$$

Matrices $\gamma_t = [\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t]$ are the state transition, control and emission matrices at time $t$. $\mathbf{Q}$ and $\mathbf{R}$ are the covariance matrices of the process and measurement noise respectively. With a starting state $\mathbf{z}_1 \sim \mathcal{N}(\mathbf{z}_1; \mathbf{0}, \mathbf{\Sigma})$, the joint probability distribution of the LGSSM is given by

$$p_\gamma(\mathbf{a}, \mathbf{z}|\mathbf{u}) = p_\gamma(\mathbf{a}|\mathbf{z})\, p_\gamma(\mathbf{z}|\mathbf{u}) = \prod_{t=1}^{T} p_{\gamma_t}(\mathbf{a}_t|\mathbf{z}_t) \cdot p(\mathbf{z}_1) \prod_{t=2}^{T} p_{\gamma_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) \ , \tag{6.3}$$

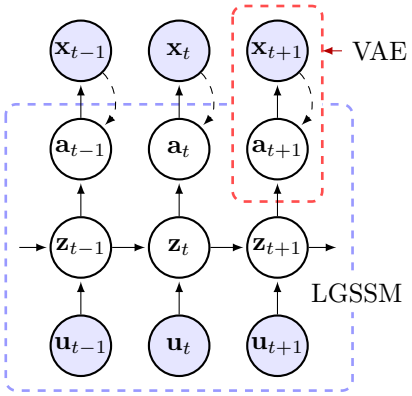



***Figure 6.1:*** *A KVAE is formed by stacking a LGSSM (dashed blue), and a VAE (dashed red). Shaded nodes denote observed variables. Solid arrows represent the generative model (with parameters $\theta$) while dashed arrows represent the VAE inference network (with parameters $\phi$).*

where $\gamma = [\gamma_1, .., \gamma_T]$. LGSSMs have very appealing properties that we wish to exploit: the filtered and smoothed posteriors $p(\mathbf{z}_t|\mathbf{a}_{1:t}, \mathbf{u}_{1:t})$ and $p(\mathbf{z}_t|\mathbf{a}, \mathbf{u})$ can be computed exactly with the classical Kalman filter and smoother algorithms, and provide a natural way to handle missing data.

**Variational auto-encoders.** A variational auto-encoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014) defines a deep generative model $p_\theta(\mathbf{x}_t, \mathbf{a}_t) = p_\theta(\mathbf{x}_t|\mathbf{a}_t)p(\mathbf{a}_t)$ for data $\mathbf{x}_t$ by introducing a latent encoding $\mathbf{a}_t$. Given a likelihood $p_\theta(\mathbf{x}_t|\mathbf{a}_t)$ and a typically Gaussian prior $p(\mathbf{a}_t)$, the posterior $p_\theta(\mathbf{a}_t|\mathbf{x}_t)$ represents a stochastic map from $\mathbf{x}_t$ to $\mathbf{a}_t$'s manifold. As this posterior is commonly analytically intractable, VAEs approximate it with a variational distribution $q_\phi(\mathbf{a}_t|\mathbf{x}_t)$ that is parameterized by $\phi$. The approximation $q_\phi$ is commonly called the recognition, encoding, or inference network.

## 6.3 Kalman Variational Auto-Encoders

The useful information that describes the movement and interplay of objects in a video typically lies in a manifold that has a smaller dimension than the number of pixels in each frame. In a video of a ball bouncing in a box, like Atari's game Pong, one could define a one-to-one mapping from each of the high-dimensional frames $\mathbf{x} = [\mathbf{x}_1, .., \mathbf{x}_T]$ into a two-dimensional latent space that represents the position of the ball on the screen. If the position was known for consecutive time steps, for a set of videos, we could learn the temporal dynamics that govern the environment. From a few new positions one might then infer where the ball will be on the screen in the future, and then imagine the environment with the ball in that position.

The *Kalman variational auto-encoder* (KVAE) is based on the notion described above. To disentangle recognition and spatial representation, a sensory input $\mathbf{x}_t$ is mapped to $\mathbf{a}_t$ (VAE), a variable on a low-dimensional manifold that encodes an object's position and other visual properties. In turn, $\mathbf{a}_t$ is used as a pseudo-observation for the dynamics model (LGSSM). $\mathbf{x}_t$ represents a frame of a video[1] $\mathbf{x} = [\mathbf{x}_1, .., \mathbf{x}_T]$ of length $T$. Each frame is encoded into a point $\mathbf{a}_t$

[1]While our main focus in this paper are videos, the same ideas could be applied more in general to any sequence

on a low-dimensional manifold, so that the KVAE contains $T$ separate VAEs that share the same decoder $p_\theta(\mathbf{x}_t|\mathbf{a}_t)$ and encoder $q_\phi(\mathbf{a}_t|\mathbf{x}_t)$, and depend on each other through a time-dependent prior over $\mathbf{a} = [\mathbf{a}_1, .., \mathbf{a}_T]$. This is illustrated in figure 6.1.

### 6.3.1 Generative model

We assume that $\mathbf{a}$ acts as a latent representation of the whole video, so that the generative model of a sequence factorizes as $p_\theta(\mathbf{x}|\mathbf{a}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{a}_t)$. In this paper $p_\theta(\mathbf{x}_t|\mathbf{a}_t)$ is a deep neural network parameterized by $\theta$, that emits either a factorized Gaussian or Bernoulli probability vector depending on the data type of $\mathbf{x}_t$. We model $\mathbf{a}$ with a LGSSM, and following (6.3), its prior distribution is

$$p_\gamma(\mathbf{a}|\mathbf{u}) = \int p_\gamma(\mathbf{a}|\mathbf{z})\, p_\gamma(\mathbf{z}|\mathbf{u})\, \mathrm{d}\mathbf{z} \ , \tag{6.4}$$

so that the joint density for the KVAE factorizes as $p(\mathbf{x}, \mathbf{a}, \mathbf{z}|\mathbf{u}) = p_\theta(\mathbf{x}|\mathbf{a})\, p_\gamma(\mathbf{a}|\mathbf{z})\, p_\gamma(\mathbf{z}|\mathbf{u})$. A LGSSM forms a convenient back-bone to a model, as the filtered and smoothed distributions $p_\gamma(\mathbf{z}_t|\mathbf{a}_{1:t}, \mathbf{u}_{1:t})$ and $p_\gamma(\mathbf{z}_t|\mathbf{a}, \mathbf{u})$ can be obtained exactly. Temporal reasoning can be done in the latent space of $\mathbf{z}_t$'s and via the latent encodings $\mathbf{a}$, and we can do long-term predictions without having to auto-regressively generate high-dimensional images $\mathbf{x}_t$. Given a few frames, and hence their encodings, one could "remain in latent space" and use the smoothed distributions to impute missing frames. Another advantage of using $\mathbf{a}$ to separate the dynamics model from $\mathbf{x}$ can be seen by considering the emission matrix $\mathbf{C}_t$. Inference in the LGSSM requires matrix inverses, and using it as a model for the prior dynamics of $\mathbf{a}_t$ allows the size of $\mathbf{C}_t$ to remain small, and not scale with the number of pixels in $\mathbf{x}_t$. While the LGSSM's process and measurement noise in (6.1) are typically formulated with full covariance matrices (Roweis and Ghahramani, 1999), we will consider them as isotropic in a KVAE, as $\mathbf{a}_t$ act as a prior in a generative model that includes these extra degrees of freedom.

What happens when a ball bounces against a wall, and the dynamics on $\mathbf{a}_t$ are not linear any more? Can we still retain a LGSSM backbone? We will incorporate nonlinearities into the LGSSM by regulating $\gamma_t$ from *outside* the exact forward-backward inference chain. We revisit this central idea at length in section 6.3.3.

### 6.3.2 Learning and inference for the KVAE

We learn $\theta$ and $\gamma$ from a set of example sequences $\{\mathbf{x}^{(n)}\}$ by maximizing the sum of their respective log likelihoods $\mathcal{L} = \sum_n \log p_{\theta\gamma}(\mathbf{x}^{(n)}|\mathbf{u}^{(n)})$ as a function of $\theta$ and $\gamma$. For simplicity in the exposition we restrict our discussion below to one sequence, and omit the sequence index $n$. The log likelihood or evidence is an intractable average over all plausible settings of $\mathbf{a}$ and $\mathbf{z}$, and exists as the denominator in Bayes' theorem when inferring the posterior $p(\mathbf{a}, \mathbf{z}|\mathbf{x}, \mathbf{u})$. A more tractable approach to both learning and inference is to introduce a variational distribution $q(\mathbf{a}, \mathbf{z}|\mathbf{x}, \mathbf{u})$ that

of high dimensional data.

approximates the posterior. The evidence lower bound (ELBO) $\mathcal{F}$ is

$$\begin{aligned}\log p(\mathbf{x}|\mathbf{u}) &= \log \int p(\mathbf{x}, \mathbf{a}, \mathbf{z}|\mathbf{u}) \\ &\geq \mathbb{E}_{q(\mathbf{a},\mathbf{z}|\mathbf{x},\mathbf{u})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{a})p_\gamma(\mathbf{a}|\mathbf{z})p_\gamma(\mathbf{z}|\mathbf{u})}{q(\mathbf{a}, \mathbf{z}|\mathbf{x}, \mathbf{u})}\right] \\ &= \mathcal{F}(\theta, \gamma, \phi) \ , \end{aligned} \tag{6.5}$$

and a sum of $\mathcal{F}$'s is maximized instead of a sum of log likelihoods. The variational distribution $q$ depends on $\phi$, but for the bound to be tight we should specify $q$ to be equal to the posterior distribution that only depends on $\theta$ and $\gamma$. Towards this aim we structure $q$ so that it incorporates the exact conditional posterior $p_\gamma(\mathbf{z}|\mathbf{a}, \mathbf{u})$, that we obtain with Kalman smoothing, as a factor of $\gamma$:

$$q(\mathbf{a}, \mathbf{z}|\mathbf{x}, \mathbf{u}) = q_\phi(\mathbf{a}|\mathbf{x})\, p_\gamma(\mathbf{z}|\mathbf{a}, \mathbf{u}) = \textstyle\prod_{t=1}^{T} q_\phi(\mathbf{a}_t|\mathbf{x}_t)\, p_\gamma(\mathbf{z}|\mathbf{a}, \mathbf{u}) \ . \tag{6.6}$$

The benefit of the LGSSM backbone is now apparent. We use a "recognition model" to encode each $\mathbf{x}_t$ using a non-linear function, after which exact smoothing is possible. In this paper $q_\phi(\mathbf{a}_t|\mathbf{x}_t)$ is a deep neural network that maps $\mathbf{x}_t$ to the mean and the diagonal covariance of a Gaussian distribution. As explained in section 6.4, this factorization allows us to deal with missing data in a principled way. Using (6.6), the ELBO in (6.5) becomes

$$\mathcal{F}(\theta, \gamma, \phi) = \mathbb{E}_{q_\phi(\mathbf{a}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{a})}{q_\phi(\mathbf{a}|\mathbf{x})} + \mathbb{E}_{p_\gamma(\mathbf{z}|\mathbf{a},\mathbf{u})}\left[\log \frac{p_\gamma(\mathbf{a}|\mathbf{z})p_\gamma(\mathbf{z}|\mathbf{u})}{p_\gamma(\mathbf{z}|\mathbf{a}, \mathbf{u})}\right]\right] \ . \tag{6.7}$$

The lower bound in (6.7) can be estimated using Monte Carlo integration with samples $\{\widetilde{\mathbf{a}}^{(i)}, \widetilde{\mathbf{z}}^{(i)}\}_{i=1}^{I}$ drawn from $q$,

$$\hat{\mathcal{F}}(\theta, \gamma, \phi) = \frac{1}{I}\sum_i \log p_\theta(\mathbf{x}|\widetilde{\mathbf{a}}^{(i)}) + \log p_\gamma(\widetilde{\mathbf{a}}^{(i)}, \widetilde{\mathbf{z}}^{(i)}|\mathbf{u}) - \log q_\phi(\widetilde{\mathbf{a}}^{(i)}|\mathbf{x}) - \log p_\gamma(\widetilde{\mathbf{z}}^{(i)}|\widetilde{\mathbf{a}}^{(i)}, \mathbf{u}) \ . \tag{6.8}$$

Note that the ratio $p_\gamma(\widetilde{\mathbf{a}}^{(i)}, \widetilde{\mathbf{z}}^{(i)}|\mathbf{u})/p_\gamma(\widetilde{\mathbf{z}}^{(i)}|\widetilde{\mathbf{a}}^{(i)}, \mathbf{u})$ in (6.8) gives $p_\gamma(\widetilde{\mathbf{a}}^{(i)}|\mathbf{u})$, but the formulation with $\{\widetilde{\mathbf{z}}^{(i)}\}$ allows stochastic gradients on $\gamma$ to also be computed. A sample from $q$ can be obtained by first sampling $\widetilde{\mathbf{a}} \sim q_\phi(\mathbf{a}|\mathbf{x})$, and using $\widetilde{\mathbf{a}}$ as an observation for the LGSSM. The posterior $p_\gamma(\mathbf{z}|\widetilde{\mathbf{a}}, \mathbf{u})$ can be tractably obtained with a Kalman smoother, and a sample $\widetilde{\mathbf{z}} \sim p_\gamma(\mathbf{z}|\widetilde{\mathbf{a}}, \mathbf{u})$ obtained from it. Parameter learning is done by *jointly* updating $\theta$, $\phi$, and $\gamma$ by maximising the ELBO on $\mathcal{L}$, which decomposes as a sum of ELBOs in (6.7), using stochastic gradient ascent and a single sample to approximate the intractable expectations.

### 6.3.3 Dynamics parameter network

The LGSSM provides a tractable way to structure $p_\gamma(\mathbf{z}|\mathbf{a}, \mathbf{u})$ into the variational approximation in (6.6). However, even in the simple case of a ball bouncing against a wall, the dynamics on $\mathbf{a}_t$ are not linear anymore. We can deal with these situations while preserving the linear dependency between consecutive states in the LGSSM, by non-linearly changing the parameters $\gamma_t$ of the model over time as a function of the latent encodings up to time $t-1$ (so that we can still define a generative model). Smoothing is still possible as the state transition matrix $\mathbf{A}_t$ and others in $\gamma_t$ do not have to be constant in order to obtain the exact posterior $p_\gamma(\mathbf{z}_t|\mathbf{a}, \mathbf{u})$.

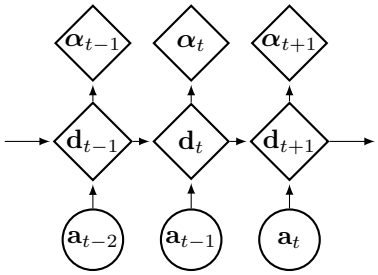



***Figure 6.2:*** *Dynamics parameter network for the KVAE.*

Recall that $\gamma_t$ describes how the latent state $\mathbf{z}_{t-1}$ changes from time $t-1$ to time $t$. In the more general setting, the changes in dynamics at time $t$ may depend on the history of the system, encoded in $\mathbf{a}_{1:t-1}$ and possibly a starting code $\mathbf{a}_0$ that can be learned from data. If, for instance, we see the ball colliding with a wall at time $t-1$, then we know that it will bounce at time $t$ and change direction. We then let $\gamma_t$ be a learnable function of $\mathbf{a}_{0:t-1}$, so that the prior in (6.3) becomes

$$p_\gamma(\mathbf{a}, \mathbf{z}|\mathbf{u}) = \prod_{t=1}^{T} p_{\gamma_t(\mathbf{a}_{0:t-1})}(\mathbf{a}_t|\mathbf{z}_t) \cdot p(\mathbf{z}_1) \prod_{t=2}^{T} p_{\gamma_t(\mathbf{a}_{0:t-1})}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) \ . \tag{6.9}$$

During inference, after all the frames are encoded in $\mathbf{a}$, the dynamics parameter network returns $\gamma = \gamma(\mathbf{a})$, the parameters of the LGSSM at all time steps. We can now use the Kalman smoothing algorithm to find the exact conditional posterior over $\mathbf{z}$, that will be used when computing the gradients of the ELBO.

In our experiments the dependence of $\gamma_t$ on $\mathbf{a}_{0:t-1}$ is modulated by a *dynamics parameter network* $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_t(\mathbf{a}_{0:t-1})$, that is implemented with a recurrent neural network with LSTM cells that takes at each time step the encoded state as input and recurses $\mathbf{d}_t = LSTM(\mathbf{a}_{t-1}, \mathbf{d}_{t-1})$ and $\boldsymbol{\alpha}_t = \text{softmax}(\mathbf{d}_t)$, as illustrated in figure 6.2. The output of the dynamics parameter network is weights that sum to one, $\sum_{k=1}^{K} \alpha_t^{(k)}(\mathbf{a}_{0:t-1}) = 1$. These weights choose and interpolate between $K$ different operating modes:

$$\mathbf{A}_t = \sum_{k=1}^{K} \alpha_t^{(k)}(\mathbf{a}_{0:t-1})\mathbf{A}^{(k)}, \quad \mathbf{B}_t = \sum_{k=1}^{K} \alpha_t^{(k)}(\mathbf{a}_{0:t-1})\mathbf{B}^{(k)}, \quad \mathbf{C}_t = \sum_{k=1}^{K} \alpha_t^{(k)}(\mathbf{a}_{0:t-1})\mathbf{C}^{(k)} \ . \tag{6.10}$$

We globally learn $K$ basic state transition, control and emission matrices $\mathbf{A}^{(k)}$, $\mathbf{B}^{(k)}$ and $\mathbf{C}^{(k)}$, and interpolate them based on information from the VAE encodings. The weighted sum can be interpreted as a soft mixture of $K$ different LGSSMs whose time-invariant matrices are combined using the time-varying weights $\boldsymbol{\alpha}_t$. In practice, each of the $K$ sets $\{\mathbf{A}^{(k)}, \mathbf{B}^{(k)}, \mathbf{C}^{(k)}\}$ models different dynamics, that will dominate when the corresponding $\alpha_t^{(k)}$ is high. The dynamics parameter network resembles the locally-linear transitions of (Karl et al., 2017; Watter et al., 2015); see section 6.6 for an in depth discussion on the differences.

## 6.4 Missing data imputation

Let $\mathbf{x}_{\text{obs}}$ be an observed subset of frames in a video sequence, for instance depicting the initial movement and final positions of a ball in a scene. From its start and end, can we imagine how

the ball reaches its final position? Autoregressive models like recurrent neural networks can only forward-generate $\mathbf{x}_t$ frame by frame, and cannot make use of the information coming from the final frames in the sequence. To impute the unobserved frames $\mathbf{x}_{\mathsf{un}}$ in the middle of the sequence, we need to do inference, not prediction.

The KVAE exploits the smoothing abilities of its LGSSM to use both the information from the past and the future when imputing missing data. In general, if $\mathbf{x} = \{\mathbf{x}_{\mathsf{obs}}, \mathbf{x}_{\mathsf{un}}\}$, the unobserved frames in $\mathbf{x}_{\mathsf{un}}$ could also appear at non-contiguous time steps, e.g. missing at random. Data can be imputed by sampling from the joint density $p(\mathbf{a}_{\mathsf{un}}, \mathbf{a}_{\mathsf{obs}}, \mathbf{z}|\mathbf{x}_{\mathsf{obs}}, \mathbf{u})$, and then generating $\mathbf{x}_{\mathsf{un}}$ from $\mathbf{a}_{\mathsf{un}}$. We factorize this distribution as

$$p(\mathbf{a}_{\mathsf{un}}, \mathbf{a}_{\mathsf{obs}}, \mathbf{z}|\mathbf{x}_{\mathsf{obs}}, \mathbf{u}) = p_\gamma(\mathbf{a}_{\mathsf{un}}|\mathbf{z})\, p_\gamma(\mathbf{z}|\mathbf{a}_{\mathsf{obs}}, \mathbf{u})\, p(\mathbf{a}_{\mathsf{obs}}|\mathbf{x}_{\mathsf{obs}}) \ , \tag{6.11}$$

and we sample from it with ancestral sampling starting from $\mathbf{x}_{\mathsf{obs}}$. Reading (6.11) from right to left, a sample from $p(\mathbf{a}_{\mathsf{obs}}|\mathbf{x}_{\mathsf{obs}})$ can be approximated with the variational distribution $q_\phi(\mathbf{a}_{\mathsf{obs}}|\mathbf{x}_{\mathsf{obs}})$. Then, if $\gamma$ is fully known, $p_\gamma(\mathbf{z}|\mathbf{a}_{\mathsf{obs}}, \mathbf{u})$ is computed with an extension to the Kalman smoothing algorithm to sequences with missing data, after which samples from $p_\gamma(\mathbf{a}_{\mathsf{un}}|\mathbf{z})$ could be readily drawn.

However, when doing missing data imputation the parameters $\gamma$ of the LGSSM are not known at all time steps. In the KVAE, each $\gamma_t$ depends on all the previous encoded states, including $\mathbf{a}_{\mathsf{un}}$, and these need to be estimated before $\gamma$ can be computed. In this paper we recursively estimate $\gamma$ in the following way. Assume that $\mathbf{x}_{1:t-1}$ is known, but not $\mathbf{x}_t$. We sample $\mathbf{a}_{1:t-1}$ from $q_\phi(\mathbf{a}_{1:t-1}|\mathbf{x}_{1:t-1})$ using the VAE, and use it to compute $\gamma_{1:t}$. The computation of $\gamma_{t+1}$ depends on $\mathbf{a}_t$, which is missing, and an estimate $\widehat{\mathbf{a}}_t$ will be used. Such an estimate can be arrived at in two steps. The filtered posterior distribution $p_\gamma(\mathbf{z}_{t-1}|\mathbf{a}_{1:t-1}, \mathbf{u}_{1:t-1})$ can be computed as it depends only on $\gamma_{1:t-1}$, and from it, we sample

$$\widehat{\mathbf{z}}_t \sim p_\gamma(\mathbf{z}_t|\mathbf{a}_{1:t-1}, \mathbf{u}_{1:t}) = \int p_{\gamma_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)\, p_\gamma(\mathbf{z}_{t-1}|\mathbf{a}_{1:t-1}, \mathbf{u}_{1:t-1})\, \mathrm{d}\mathbf{z}_{t-1} \tag{6.12}$$

and sample $\widehat{\mathbf{a}}_t$ from the predictive distribution of $\mathbf{a}_t$,

$$\widehat{\mathbf{a}}_t \sim p_\gamma(\mathbf{a}_t|\mathbf{a}_{1:t-1}, \mathbf{u}_{1:t}) = \int p_{\gamma_t}(\mathbf{a}_t|\mathbf{z}_t)\, p_\gamma(\mathbf{z}_t|\mathbf{a}_{1:t-1}, \mathbf{u}_{1:t})\, \mathrm{d}\mathbf{z}_t \approx p_{\gamma_t}(\mathbf{a}_t|\widehat{\mathbf{z}}_t) \ . \tag{6.13}$$

The parameters of the LGSSM at time $t+1$ are then estimated as $\gamma_{t+1}([\mathbf{a}_{0:t-1}, \widehat{\mathbf{a}}_t])$. The same procedure is repeated at the next time step if $\mathbf{x}_{t+1}$ is missing, otherwise $\mathbf{a}_{t+1}$ is drawn from the VAE. After the forward pass through the sequence, where we estimate $\gamma$ and compute the filtered posterior for $\mathbf{z}$, the Kalman smoother's backwards pass computes the smoothed posterior. While the smoothed posterior distribution is not exact, as it relies on the estimate of $\gamma$ obtained during the forward pass, it improves data imputation by using information coming from the whole sequence; see section 6.5 for an experimental illustration.

## 6.5 Experiments

We motivated the KVAE with an example of a bouncing ball, and use it here to demonstrate the model's ability to separately learn a recognition and dynamics model from video, and use it to

impute missing data. To draw a comparison with deep variational Bayes filters (DVBFs) (Karl et al., 2017), we apply the KVAE to (Karl et al., 2017)'s pendulum example. We further apply the model to a number of environments with different properties to demonstrate its generalizability. All models are trained end-to-end with stochastic gradient descent. Using the control input $\mathbf{u}_t$ in (6.1) we can inform the model of known quantities such as external forces, as will be done in the pendulum experiment. In all the other experiments, we omit such information and train the models fully unsupervised from the videos only. Further implementation details can be found in the supplementary material (appendix 6.8) and in the Tensorflow (Abadi et al., 2015) code released at github.com/simonkamronn/kvae.

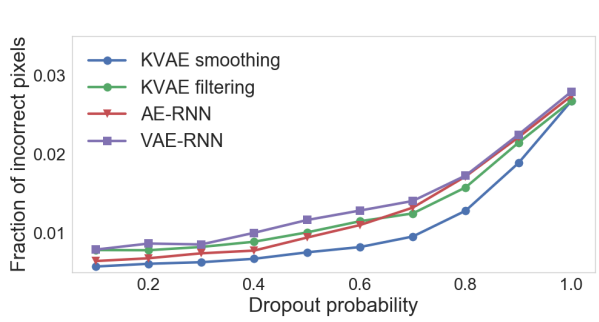
### 6.5.1 Bouncing ball

We simulate 5000 sequences of 20 time steps each of a ball moving in a two-dimensional box, where each video frame is a 32x32 binary image. A video sequence is visualised as a single image in figure 6.4d, with the ball's darkening color reflecting the incremental frame index. In this set-up the initial position and velocity are randomly sampled. No forces are applied to the ball, except for the fully elastic collisions with the walls. The minimum number of latent dimensions that the KVAE requires to model the ball's dynamics are $\mathbf{a}_t \in \mathbb{R}^2$ and $\mathbf{z}_t \in \mathbb{R}^4$, as at the very least the ball's position in the box's 2d plane has to be encoded in $\mathbf{a}_t$, and $\mathbf{z}_t$ has to encode the ball's position and velocity. The model's flexibility increases with more latent dimensions, but we choose these settings for the sake of interpretable visualisations. The dynamics parameter network uses $K = 3$ to interpolate three modes, a constant velocity, and two non-linear interactions with the horizontal and vertical walls.

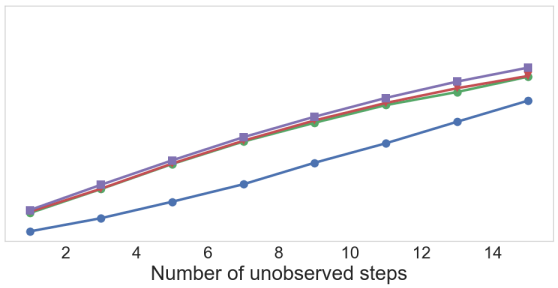
We compare the generation and imputation performance of the KVAE with two recurrent neural network (RNN) models that are based on the same auto-encoding (AE) architecture as the KVAE and are modifications of methods from the literature to be better suited to the bouncing ball experiments.[2] In the *AE-RNN*, inspired by the architecture from (Srivastava et al., 2015), a pretrained convolutional auto-encoder, identical to the one used for the KVAE, feeds the encodings to an LSTM network (Hochreiter and Schmidhuber, 1997). During training the LSTM predicts the next encoding in the sequence and during generation we use the previous output as input to the current step. For data imputation the LSTM either receives the previous output or, if available, the encoding of the observed frame (similarly to filtering in the KVAE). The *VAE-RNN* is identical to the AE-RNN except that uses a VAE instead of an AE, similarly to the model from (Chung et al., 2015).

**Figure 6.3a** shows how well missing frames are imputed in terms of the average fraction of incorrectly guessed pixels. In it, the first 4 frames are observed (to initialize the models) after which the next 16 frames are dropped at random with varying probabilities. We then impute the missing frames by doing filtering and smoothing with the KVAE. We see in figure 6.3a that it is beneficial to utilize information from the whole sequence (even the future observed frames), and a KVAE with smoothing outperforms all competing methods. Notice that dropout probability 1 corresponds to pure generation from the models. **Figure 6.3b** repeats this experiment, but makes it more challenging by removing an increasing number of *consecutive* frames from the

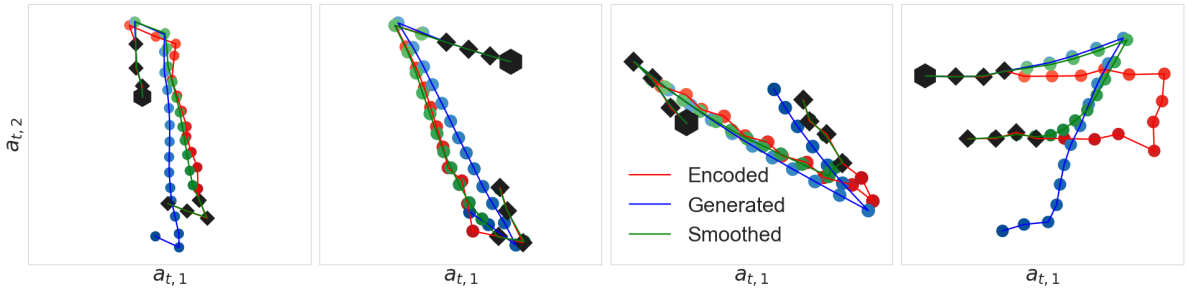
[2] We also experimented with the SRNN model from (Fraccaro et al., 2016c) as it can do smoothing. However, the model is probably too complex for the task in hand, and we could not make it learn good dynamics.




*(a) Frames $\mathbf{x}_t$ missing completely at random.*




*(b) Frames $\mathbf{x}_t$ missing in the middle of the sequence.*



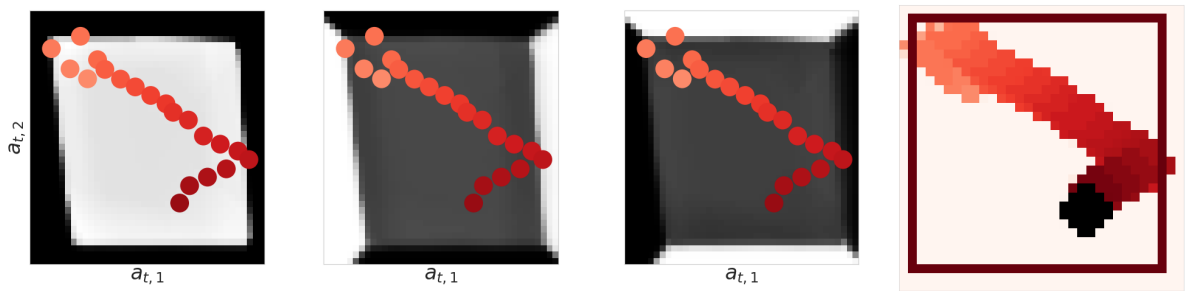

*(c) Comparison of encoded (ground truth), generated and smoothed trajectories of a KVAE in the latent space $\mathbf{a}$. The black squares illustrate observed samples and the hexagons indicate the initial state. Notice that the $\mathbf{a}_t$'s lie on a manifold that can be rotated and stretched to align with the frames of the video.*

***Figure 6.3:*** *Missing data imputation results.*

middle of the sequence ($T = 20$). In this case the ability to encode information coming from the future into the posterior distribution is highly beneficial, and smoothing imputes frames much better than the other methods. **Figure 6.3c** graphically illustrates figure 6.3b. We plot three trajectories over $\mathbf{a}_t$-encodings. The *generated* trajectories were obtained after initializing the KVAE model with 4 initial frames, while the *smoothed* trajectories also incorporated encodings from the last 4 frames of the sequence. The *encoded* trajectories were obtained with no missing data, and are therefore considered as ground truth. In the first three plots in figure 6.3c, we see that the backwards recursion of the Kalman smoother corrects the trajectory obtained with generation in the forward pass. However, in the fourth plot, the poor trajectory that is obtained during the forward generation step, makes smoothing unable to follow the ground truth.

The smoothing capabilities of KVAEs make it also possible to train it with up to 40% of missing data with minor losses in performance (appendix 6.10 in the supplementary material). Links to videos of the imputation results and long-term generation from the models can be found in appendix 6.9 and at sites.google.com/view/kvae.

**Understanding the dynamics parameter network.** In our experiments the dynamics parameter network $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_t(\mathbf{a}_{0:t-1})$ is an LSTM network, but we could also parameterize it with any differentiable function of $\mathbf{a}_{0:t-1}$ (see appendix 6.11 in the supplementary material for a comparison of various architectures). When using a multi-layer perceptron (MLP) that depends on the previous encoding as mixture network, i.e. $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_t(\mathbf{a}_{t-1})$, figure 6.4 illustrates how the network chooses the mixture of learned dynamics. We see that the model has correctly learned




*(a)* $k = 1$ *(b)* $k = 2$ *(c)* $k = 3$ *(d) Reconstruction of* $\mathbf{x}$

***Figure 6.4:*** *A visualisation of the dynamics parameter network* $\alpha_t^{(k)}(\mathbf{a}_{t-1})$ *for* $K = 3$*, as a function of* $\mathbf{a}_{t-1}$*. The three* $\alpha_t^{(k)}$*'s sum to one at every point in the encoded space. The greyscale backgrounds in* ***a)*** *to* ***c)*** *correspond to the intensity of the weights* $\alpha_t^{(k)}$*, with white indicating a weight of one in the dynamics parameter network's output. Overlaid on them is the full latent encoding* $\mathbf{a}$*.* ***d)*** *shows the reconstructed frames of the video as one image.*

| Model | Test ELBO |
|---|---|
| KVAE (CNN) | 810.08 |
| KVAE (MLP) | 807.02 |
| DVBF | 798.56 |
| DMM | 784.70 |

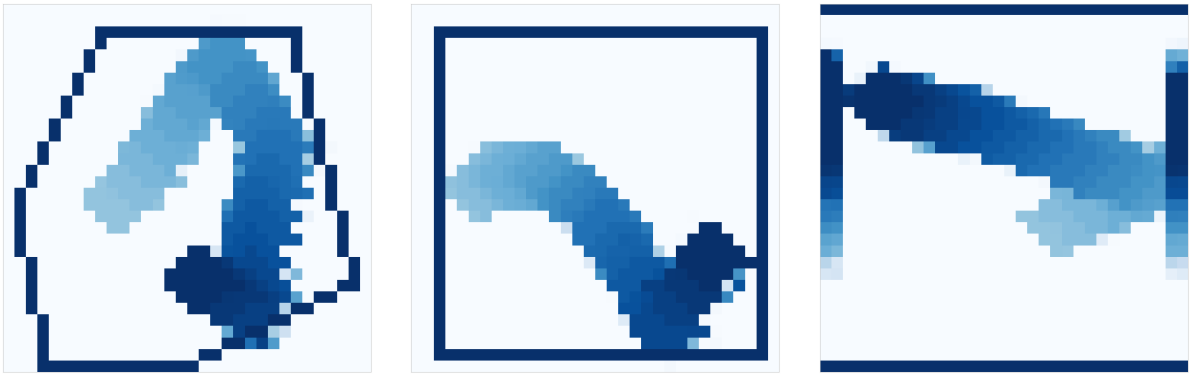
***Table 6.1:*** *Pendulum experiment.*

to choose a transition that maintains a constant velocity in the center ($k = 1$), reverses the horizontal velocity when in proximity of the left and right wall ($k = 2$), reverses the vertical velocity when close to the top and bottom ($k = 3$).

### 6.5.2 Pendulum experiment

We test the KVAE on the experiment of a dynamic torque-controlled pendulum used in (Karl et al., 2017). Training, validation and test set are formed by 500 sequences of 15 frames of 16x16 pixels. We use a KVAE with $\mathbf{a}_t \in \mathbb{R}^2$, $\mathbf{z}_t \in \mathbb{R}^3$ and $K = 2$, and try two different encoder-decoder architectures for the VAE, one using a MLP and one using a convolutional neural network (CNN). We compare the performaces of the KVAE to DVBFs (Karl et al., 2017) and deep Markov models[3] (DMM) (Krishnan et al., 2017), non-linear SSMs parameterized by deep neural networks whose intractable posterior distribution is approximated with an inference network. In table 6.1 we see that the KVAE outperforms both models in terms of ELBO on a test set, showing that for the task in hand it is preferable to use a model with simpler dynamics but exact posterior inference.

[3]Deep Markov models were previously referred to as deep Kalman filters.



*(a) Irregular polygon.* *(b) Box with gravity.* *(c) Pong-like environment.*

***Figure 6.5:*** *Generations from the KVAE trained on different environments. The videos are shown as single images, with color intensity representing the incremental sequence index $t$. In the simulation that resembles Atari's Pong game, the movement of the two paddles (left and right) is also visible.*

### 6.5.3 Other environments

To test how well the KVAE adapts to different environments, we trained it end-to-end on videos of (i) a ball bouncing between walls that form an irregular polygon, (ii) a ball bouncing in a box and subject to gravity, (iii) a Pong-like environment where the paddles follow the vertical position of the ball to make it stay in the frame at all times. Figure 6.5 shows that the KVAE learns the dynamics of all three environments, and generates realistic-looking trajectories. We repeat the imputation experiments of figures 6.3a and 6.3b for these environments in the supplementary material (appendix 6.12), where we see that KVAEs outperform alternative models.

## 6.6 Related work

Recent progress in unsupervised learning of high dimensional sequences is found in a plethora of both deterministic and probabilistic generative models. The VAE framework is a common work-horse in the stable of probabilistic inference methods, and it is extended to the temporal setting by (Archer et al., 2015; Chung et al., 2015; Fraccaro et al., 2016c; Karl et al., 2017; Krishnan et al., 2017). In particular, deep neural networks can parameterize the transition and emission distributions of different variants of deep state-space models (Fraccaro et al., 2016c; Karl et al., 2017; Krishnan et al., 2017). In these extensions, inference networks define a variational approximation to the intractable posterior distribution of the latent states at each time step. For the tasks in section 6.5, it is preferable to use the KVAE's simpler temporal model with an exact (conditional) posterior distribution than a highly non-linear model where the posterior needs to be approximated. A different combination of VAEs and probabilistic graphical models has been explored in (Johnson et al., 2016), which defines a general class of models where inference is performed with message passing algorithms that use deep neural networks to map the observations to conjugate graphical model potentials.

In classical non-linear extensions of the LGSSM like the extended Kalman filter and in the

locally-linear dynamics of (Karl et al., 2017; Watter et al., 2015), the transition matrices at time $t$ have a non-linear dependence on $\mathbf{z}_{t-1}$. The KVAE's approach is different: by introducing the latent encodings $\mathbf{a}_t$ and making $\gamma_t$ depend on $\mathbf{a}_{1:t-1}$, the *linear* dependency between consecutive states of $\mathbf{z}$ is preserved, so that the exact smoothed posterior can be computed given $\mathbf{a}$, and used to perform missing data imputation. LGSSM with dynamic parameterization have been used for large-scale demand forecasting in (Seeger et al., 2016). (Linderman et al., 2017) introduces recurrent switching linear dynamical systems, that combine deep learning techniques and switching Kalman filters (Murphy, 1998) to model low-dimensional time series. (Haarnoja et al., 2016) introduces a *discriminative* approach to estimate the low-dimensional state of a LGSSM from input images. The resulting model is reminiscent of a KVAE with no decoding step, and is therefore not suited for unsupervised learning and video generation. Recent work in the non-sequential setting has focused on disentangling basic visual concepts in an image (Higgins et al., 2017a). (Gao et al., 2016) models neural activity by finding a non-linear embedding of a neural time series into a LGSSM.

Great strides have been made in the reinforcement learning community to model how environments evolve in response to action (Chiappa et al., 2017; Oh et al., 2015; Patraucean et al., 2015; Sun et al., 2016; Wahlström et al., 2015). In similar spirit to this paper, (Wahlström et al., 2015) extracts a latent representation from a PCA representation of the frames where controls can be applied. (Chiappa et al., 2017) introduces action-conditional dynamics parameterized with LSTMs and, as for the KVAE, a computationally efficient procedure to make long term predictions without generating high dimensional images at each time step. As autoregressive models, (Srivastava et al., 2015) develops a sequence to sequence model of video representations that uses LSTMs to define both the encoder and the decoder. (Finn et al., 2016) develops an action-conditioned video prediction model of the motion of a robot arm using convolutional LSTMs that models the change in pixel values between two consecutive frames.

While the focus in this work is to define a generative model for *high dimensional* videos of simple physical systems, several recent works have combined physical models of the world with deep learning to learn the dynamics of objects in more complex but *low-dimensional* environments (Battaglia et al., 2016; Chang et al., 2017; Fragkiadaki et al., 2016; Wu et al., 2015).

## 6.7 Conclusion

The KVAE, a model for unsupervised learning of high-dimensional videos, was introduced in this paper. It disentangles an object's latent representation $\mathbf{a}_t$ from a latent state $\mathbf{z}_t$ that describes its dynamics, and can be learned end-to-end from raw video. Because the exact (conditional) smoothed posterior distribution over the states of the LGSSM can be computed, one generally sees a marked improvement in inference and missing data imputation over methods that don't have this property. A desirable property of disentangling the two latent representations is that temporal reasoning, and possibly planning, could be done in the latent space. As a proof of concept, we have been deliberate in focussing our exposition to videos of static worlds that contain a few moving objects, and leave extensions of the model to real world videos or sequences coming from an agent exploring its environment to future work.

# Supplementary material

## 6.8 Experimental details

We will describe here some of the most important experimental details. The rest of the details can be found in the code at github.com/simonkamronn/kvae.

**Data generation.** All the videos were generated using the physics engine Pymunk. We generated 5000 videos for training and 1000 for testing.

**Encoder/Decoder architecture for the KVAE.** As we only use image-based observations, the encoder is fixed to a three layer convolutional neural network with 32 units in each layer, kernel-size of 3x3, stride of 2, and ReLU activations. The decoder is an equally sized network using the Sub-Pixel(Shi et al., 2016) procedure for deconvolution. In the pendulum experiment however we also test MLPs.

**Optimization.** As optimizer we use ADAM (Kingma and Ba, 2014) with an initial learning rate of 0.007 and an exponential decay scheme with a rate of 0.85 every 20 epochs. Training one epoch takes 55 seconds on an NVIDIA Titan X and the model converges in roughly 80 epochs.

**Training tricks for end-to-end learning.** The biggest challenge of this optimization problem is how to avoid poor local minima, for example where all the focus is given to the reconstruction term, at the expense of the prior dynamics given by the LGSSM. To achieve a quick convergence in all the experiments we found it helpful to

- downweight the reconstruction term from of VAEs during training, that is scaled by 0.3. By doing this, we can in fact help the model to focus on learning the temporal dynamics.
- learn for the first few epochs only the the VAE parameters $\theta$ and $\phi$ and the globally learned matrices $\mathbf{A}^{(k)}$, $\mathbf{B}^{(k)}$ and $\mathbf{C}^{(k)}$, but not the parameters of the dynamics parameter network $\boldsymbol{\alpha}_t(\mathbf{a}_{0:t-1})$. After this phase, all parameters are learned jointly. This allows the model to first learn good VAE embeddings and the scale of the prior, and then learn how to utilize the $K$ different dynamics.

**Choice of hyperparameters for the LGSSM.** In most of the experiments we used $\mathbf{a}_t \in \mathbb{R}^2$, $\mathbf{z}_t \in \mathbb{R}^4$ and $K = 3$. In the *gravity* experiments we used however $\mathbf{z}_t \in \mathbb{R}^5$ as the model has no controls applied to it and needs to be able to learn a bias term due to the presence of the external force of gravity. The *polygon* experiments uses $K = 7$ as it needs to learn more complex dynamics. In general, we did not find difficult to tune the parameters of the KVAE, as the model can learn to prune unused components (if flexible enough).

## 6.9 Videos

Videos are generated from all models by initializing with 4 frames and then sampling. The *filtering* and *smoothing* versions are allowed to observe part of the sequence depending on the masking scheme. All the *filtering* and *smoothing* videos are generated from sequences applied with a random mask with a masking probability of 80% (as in figure 6.3a) except for the videos with the suffix *consecutive* in which only the first and last 4 frames are observed (as in figure 6.3b). Only the KVAE models have *smoothing* videos. For the bouncing ball experiment (named *box* in the attached folder), we also show the videos from a model trained with 40% missing data.

In most videos the black ball is the ground truth, and the red is the one generated from the model, except for the ones marked *long_generation* in which the true sequence is not shown.

Videos are available from Google Drive and the website sites.google.com/view/kvae.

## 6.10 Training with missing data.

The smoothed posterior described in section 6.4 can also be used to train the KVAE with missing data. In this case, we only need to modify the ELBO by masking the contribution of the missing data points in the joint probability distribution and variational approximation:

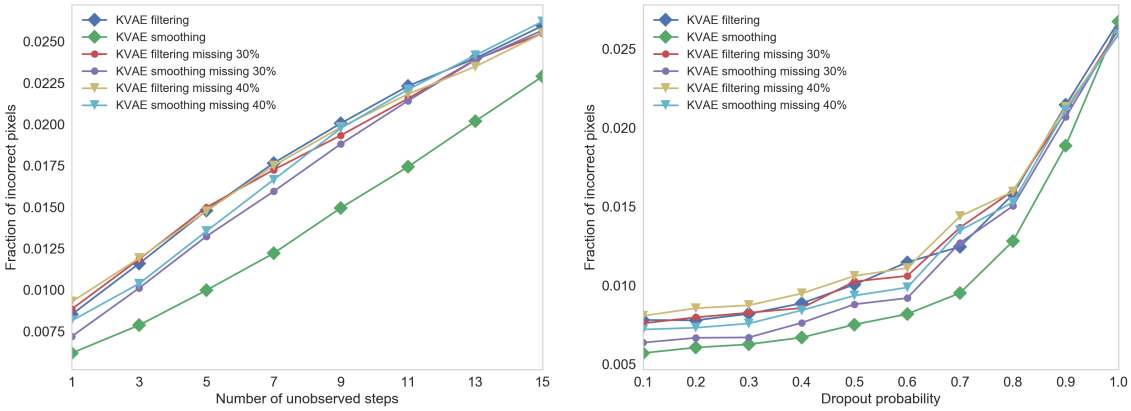
$$p(\mathbf{x}, \mathbf{a}, \mathbf{z}, \mathbf{u}) = p(\mathbf{z}_1) \prod_{t=2}^{T} p_{\gamma_t}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) \prod_{t=1}^{T} p_{\gamma_t}(\mathbf{a}_t|\mathbf{z}_t)^{\mathcal{I}_t} \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{a}_t)^{\mathcal{I}_t}$$
$$q_\phi(\mathbf{a}|\mathbf{x}) = \prod_{t=1}^{T} q_\phi(\mathbf{a}_t|\mathbf{x}_t)^{\mathcal{I}_t} \ ,$$

where $\mathcal{I}_t$ is 0 if the data point is missing, 1 otherwise. Figure 6.6 illustrates a slight degradation in performance when training with respectively 30% and 40% missing data but, remarkably, the accuracy is still better when using smoothing in these conditions than with filtering with all training data available.

## 6.11 Dynamics parameter network architecture

As the $\alpha$-network governs the non-linear dynamics, it has a significant impact on the modelling capabilities. Here we list the architectural choices considered:

- **MLP** with two hidden layers.
- **Recurrent Neural Networks** with LSTM units.
- **'First in, first out memory' (FIFO) MLP** with access to 5 time steps.




*(a)* *(b)*

***Figure 6.6:*** *Training with missing data*

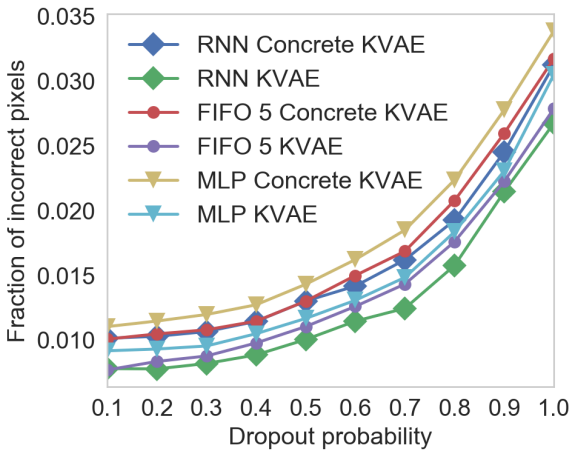
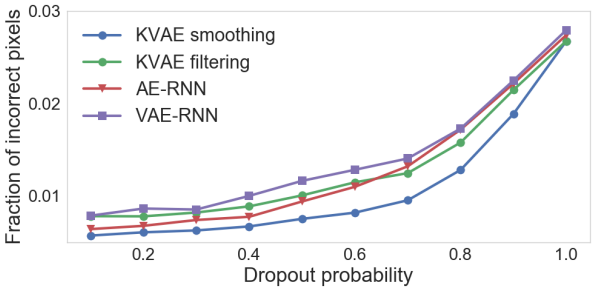



***Figure 6.7:*** *Comparison of modelling choices wrt. the $\alpha$-network*

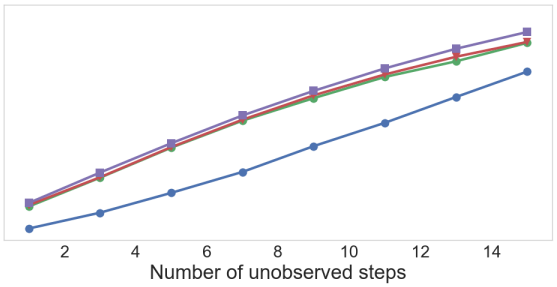
In all cases, we can also model $\alpha$ as an (approximate) discrete random variable using the the Concrete distribution (Maddison et al., 2017b; Jang et al., 2016). In this case we can recover an approximation to the switching Kalman filter(Murphy, 1998).

In figure 6.7 the different choices are tested against each other on the bouncing ball data. In this case all the alternative choices result in poorer performances than the LSTM chosen for all the other experiments. We believe that LSTMs are able to better model the discretization errors coming from the collisions and the 32x32 rendering of the trajectories computed by the physics engine.

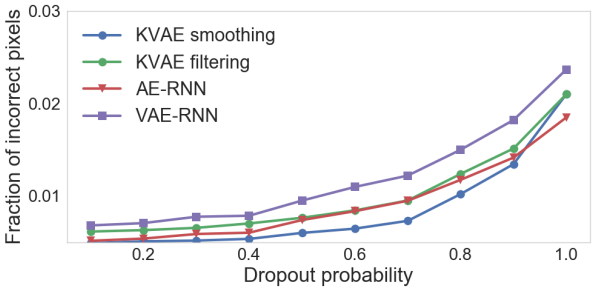
## 6.12 Imputation in all environments



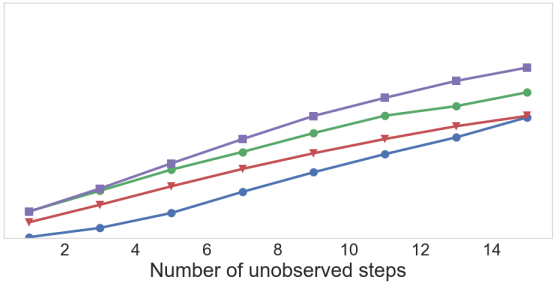

*(a) Bouncing ball - Frames $\mathbf{x}_t$ missing randomly.*



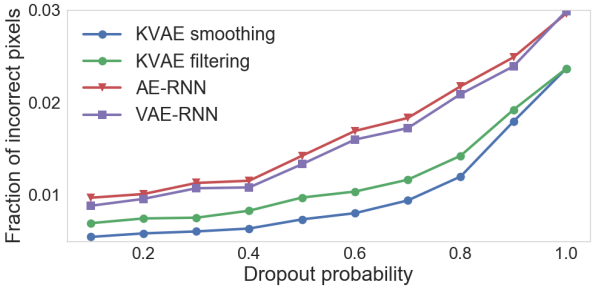

*(b) Bouncing ball - Frames $\mathbf{x}_t$ missing in the middle*



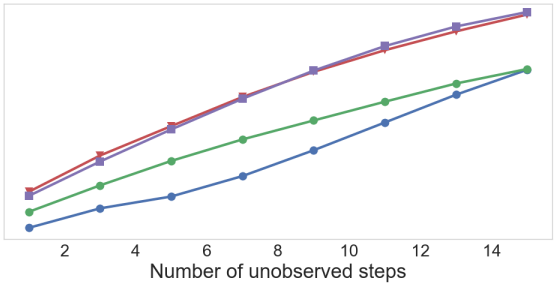

*(c) Gravity - Frames $\mathbf{x}_t$ missing randomly.*



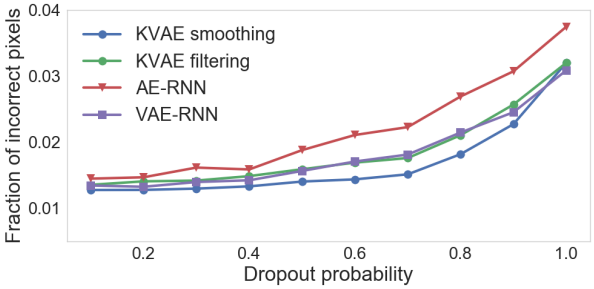

*(d) Gravity - Frames $\mathbf{x}_t$ missing in the middle*



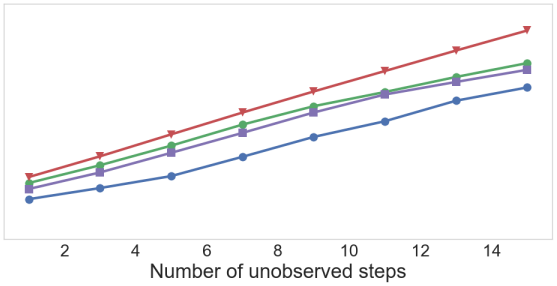

*(e) Polygon - Frames $\mathbf{x}_t$ missing randomly.*




*(f) Polygon - Frames $\mathbf{x}_t$ missing in the middle*




*(g) Pong - Frames $\mathbf{x}_t$ missing randomly.*




*(h) Pong - Frames $\mathbf{x}_t$ missing in the middle*

***Figure 6.8:*** *Imputation results for all environments*

Chapter 7

# Generative Temporal Models with Spatial Memory for Partially Observed Environments

Marco Fraccaro[a] · Danilo Rezende[b] · Yori Zwols[b] · Alexander Pritzel[b] · Ali Eslami[b] · Fabio Viola[b]

[a]DTU Compute, Technical University of Denmark, Denmark

[b]Google DeepMind, United Kingdom

**Publication Status:** Published in the Proceedings of the 35th International Conference on Machine Learning, ICML 2018. Selected for long talk. Work done during an internship at Google DeepMind.

**Abstract:** In model-based reinforcement learning, generative and temporal models of environments can be leveraged to boost agent performance, either by tuning the agent's representations during training or via use as part of an explicit planning mechanism. However, their application in practice has been limited to simplistic environments, due to the difficulty of training such models in larger, potentially partially-observed and 3D environments. In this work we introduce a novel action-conditioned generative model of such challenging environments. The model features a non-parametric spatial memory system in which we store learned, disentangled representations of the environment. Low-dimensional spatial updates are computed using a state-space model that makes use of knowledge on the prior dynamics of the moving agent, and high-dimensional visual observations are modelled with a Variational Auto-Encoder. The result is a scalable architecture capable of performing coherent predictions over hundreds of time steps across a range of partially observed 2D and 3D environments.

## 7.1 Introduction

Consider a setup in which an agent walks and observes an environment (e.g., a three-dimensional maze) for hundreds of time steps, and is then asked to predict subsequent observations given a sequence of actions. This is a challenging task, as it requires the ability to first remember the visual observations and the position in which they were observed in the environment, and secondly to predict where a possibly long sequence of actions would bring the agent in the environment. Building models that can solve this problem can be useful for model-based reinforcement learning involving spatial tasks that require long-term memories and other spatial downstream goals (Sutton, 1990; Deisenroth and Rasmussen, 2011; Levine and Abbeel, 2014; Watter et al., 2015; Wahlström et al., 2015; Lenz et al., 2015; Higgins et al., 2017b; Finn and Levine, 2017). This requires however agents that are able to remember the past over hundreds of steps, that know both where they are in the environment and how each action changes their position, and that can coherently predict hundreds of steps into the future. Therefore the main focus of this work is to develop an action-conditioned generative model that is able to memorize all the required information while exploring the environment and successively use it in the prediction phase for long-term generation of high-dimensional visual observations.

Recently, several powerful generative models for sequential data have been proposed in a wide range of applications, such as modelling speech, handwriting, polyphonic music and videos (Chung et al., 2015; Fraccaro et al., 2016c; Oh et al., 2015; Chiappa et al., 2017). They build on recurrent neural architectures such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) or Gated Recurrent Units (GRU) (Chung et al., 2014), that use an internal state vector to perform computations and store the long-term information needed when making predictions. Since the number of parameters in these models scales quadratically with the dimensionality of the state vector, they are not suitable for applications that require high memory capacity, such as the one considered in this paper. The high dimensional state vector needed to be able to memorize the hundreds of time steps in which the agent has visited the whole environment, make these models in practice both very slow and hard to train. An alternative approach is to use an external memory architecture, for storing of large amount of information while drastically reducing the number of parameters with respect to models with similar memory capacity that build on LSTMs or GRUs. Gemici et al., (2017) present a general architecture for generative temporal models with external memory, and test four different types of memories that are dynamically updated at each time step (Graves et al., 2014; Graves et al., 2016; Santoro et al., 2016). They focus on differentiable addressing mechanisms for memory storage and retrieval (soft-attention), that are based on deep neural networks that learn to write information to the memory and read from it. While this approach is very general and can be used to model complex long-term temporal dependencies in a wide range of applications, it has not been successful in modeling the data coming from an agent freely moving in a 3d maze, even for a single room [private communications with the authors of (Gemici et al., 2017)].

To define a scalable model capable of exploring larger environments and coherently predicting hundreds of time steps in the future, in this work we build a *spatial memory* architecture that exploits some knowledge of the specific structure of the problem in consideration. In particular, at each time step we split the internal latent representation of the system in to two separate vectors, a low-dimensional one that encodes the position of the agent in the environment and a high dimensional one that encodes what the agent is seeing. We model the low dimensional dynamics

of the agent with a state-space model in which we encode prior information on the physical principles that govern the agent's movements, and learn a higher dimensional latent representation of the visual input (the frames from the environment) with a Variational Auto-Encoder (Kingma and Welling, 2014; Rezende et al., 2014). While exploring the environment, at each time step we store the position of the agent and the corresponding visual information in a Differentiable Neural Dictionary (DND) (Pritzel et al., 2017), a scalable non-parametric memory developed for episodic control. The resulting model is able to coherently generate hundreds of time steps into the future in simulated 3D environments, by retrieving at each time step the observations stored in memory that were collected when passing in nearby positions during the exploration phase. Making predictions with our model is scalable because of the efficient rollouts in a low dimensional space made possible by the state-space assumption and the efficient retrieval of the necessary information from DND. The proposed model can be trained end-to-end on videos with corresponding action sequences of agents walking in an environment. Importantly, unlike the work in (Gemici et al., 2017) we do not need to learn a complex memory addressing mechanisms, as in our model the DND represents a non-parametric component where we store encodings of the positions and visual information that are learned from the data in an unsupervised way.

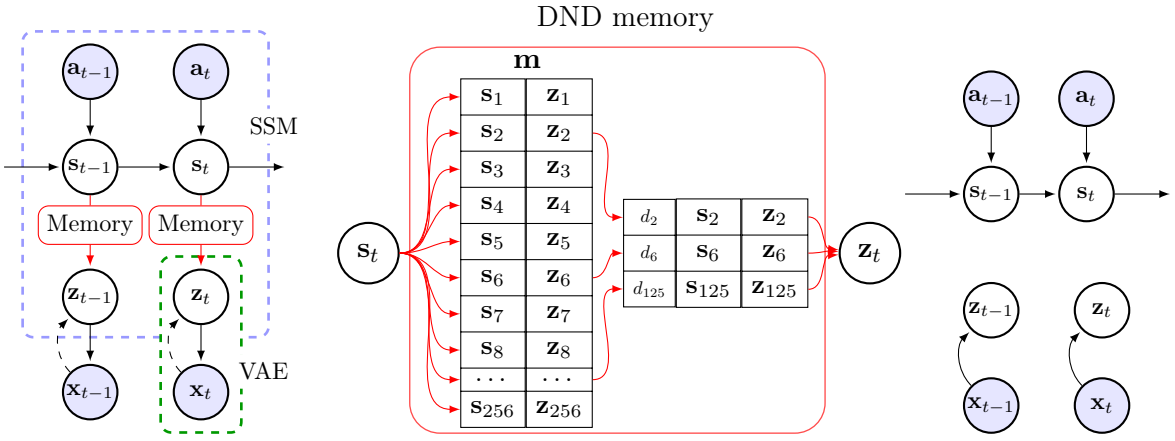
## 7.2 Background

We now provide a brief overview of the building blocks for the model introduced in section 7.3, namely variational auto-encoders, the DND memory and state-space models.

**Variational auto-encoders.** Variational auto-encoders (VAEs) (Kingma and Welling, 2014; Rezende et al., 2014) define a generative model for high-dimensional data $\mathbf{x}_t$ by introducing a latent state $\mathbf{z}_t$. The joint probability distribution $p_\theta(\mathbf{x}_t, \mathbf{z}_t)$ is factorized as $p_\theta(\mathbf{x}_t, \mathbf{z}_t) = p_\theta(\mathbf{x}_t|\mathbf{z}_t)p(\mathbf{z}_t)$, where $p(\mathbf{z}_t)$ is the prior of the latent state and the *decoder* $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$ defines a mapping using deep neural networks parameterized by $\theta$ from the states $\mathbf{z}_t$ to the data $\mathbf{x}_t$. In a VAE, the intractable posterior distribution over the latent states is approximated using the variational distribution $q_\phi(\mathbf{z}_t|\mathbf{x}_t)$, also known as the *encoder* or *inference network*. The parameters $\theta$ and $\phi$ of the decoder and the encoder, respectively, are learned jointly by maximizing the Evidence Lower Bound (ELBO) with stochastic gradient ascent.

**DND memory.** The Differentiable Neural Dictionary (DND) is a scalable, non-parametric memory module first introduced in Reinforcement Learning (RL) to allow agents to store and retrieve their experiences of an environment (Pritzel et al., 2017). The *write* operation consists of inserting (key, value) pairs into the memory; similarly to a dictionary, this associates a value to each key. Given a query key, we can then *read* from the memory by finding among the keys stored in the DND the nearest neighbours to the query key and returning the corresponding values. The DND can be used in applications that require very large memories, since the nearest-neighbour search can be efficiently approximated using space-partitioning data structures, such as kd-trees (Bentley, 1975).

**State-space models.** State-space models (SSM) are a class of probabilistic graphical models widely used in the temporal setting to model sequences of vectors $\mathbf{z}_{1:T} = [\mathbf{z}_1, .., \mathbf{z}_T]$ conditioned on some actions $\mathbf{a}_{1:T} = [\mathbf{a}_1, .., \mathbf{a}_T]$. SSMs introduce at each time step a continuous stochastic variable $\mathbf{s}_t$, used as a latent representation of the state of the system. The temporal dynamics of




*(a)* *Generative model of the GTM-SM. Red arrows represent dependencies on the DND memory. Dashed arrows represent the VAE inference network.*

*(b)* *The dependence of $\mathbf{z}_t$ on $\mathbf{s}_t$ and the DND memory $\mathbf{m}$ in the VAE prior $p(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$. This corresponds to the red rectangle in Figure 7.1a. Here, the memory contains the first $\tau = 256$ time steps of the video, and we retrieve the $K = 3$ closest neighbours to $\mathbf{s}_t$ ($d$ is the distance).*

*(c)* *Inference network for the GTM-SM.*

***Figure 7.1:*** *Generative Temporal Model with Spatial Memory*

the system are described by the *transition density* $p(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t)$ of the SSM, that defines how to update the state at time $t$ given the previous state $\mathbf{s}_{t-1}$ and the current action $\mathbf{a}_t$. The output variable $\mathbf{z}_t$ depends on the state $\mathbf{s}_t$ through the *emission density* $p(\mathbf{z}_t|\mathbf{s}_t)$.

## 7.3 Model

An important component of model-based reinforcement learning is the ability to plan many steps ahead in time leveraging previous experiences (Sutton, 1990; Racanière et al., 2017). This requires agents that can remember the past and use it to predict what may happen in the future given certain actions. With this purpose in mind, we define an action-conditioned generative model with memory, that can be used within RL agents for model-based planning.

The input of our model consists of $T$-step videos with corresponding action sequences, generated by an agent acting in an environment. We split each sequence of $T$ time steps into two parts, corresponding to two different model phases:

1. *Memorization phase.* For $t = 1, .., \tau$, the model receives at each time step a frame $\mathbf{x}_t$ and action $\mathbf{a}_t$ (e.g. move forwards/backwards, rotate left/right) that led to it. In this phase, the model has to store in memory all the information needed in the following prediction phase. During this phase the agent sees most of the environment (but from a restricted set of viewpoints), in order to give the model sufficient information to make accurate predictions in the subsequent phase.

2. *Prediction phase.* For $t = \tau + 1, .., T$, the model receives the actions $\mathbf{a}_{\tau+1:T}$ that move the

data-generating agent across the previously explored environment (although perhaps viewed from a different angle) and needs to predict the observations $\mathbf{x}_{\tau+1:T}$ using the information about the past that is stored in the memory.

Storing *what* the agent sees at each time step is not sufficient: in order to retrieve the correct information from memory when returning to the same location during the prediction phase, we also need to store *where* the agent is. The location of the agent is a latent variable that can be inferred given the actions as explained in the rest of this section.

As shown in Figure 7.1a, in a *Generative Temporal Model with Spatial Memory (GTM-SM)* we introduce two sets of latent variables that disentangle visual and dynamics information, similarly to (Fraccaro et al., 2017). At each time step we have a VAE whose latent state $\mathbf{z}_t$ is an encoding of the frame of the video and therefore captures the visual information. The priors of the VAEs are temporally dependent through their dependence on the states $\mathbf{s}_t$ of the SSM, a latent representation of the location of the agent in the environment. The transition density of the SSM is used to include prior knowledge on the environment dynamics, i.e. the underlying physics.

During the initial memorization phase, the GTM-SM infers the states $\mathbf{s}_{1:\tau}$ of the agent (i.e. the position) and the frame encodings $\mathbf{z}_{1:\tau}$, and stores these $(\mathbf{s}_i, \mathbf{z}_i)$ pairs as (key, value) in the DND memory. Probabilistically, we can view this as inserting an approximation of the intractable the posterior $p(\mathbf{s}_{1:\tau}, \mathbf{z}_{1:\tau} | \mathbf{x}_{1:\tau}, \mathbf{a}_{1:\tau})$ into the DND; see Section 7.3.3 for details. As the latent variables are stochastic, in practice we store in memory the sufficient statistics of the distribution (e.g. the mean and variance in the case of Gaussian variables). To keep the notation simple, we will refer to the information in the DND by the name of the random variable (as done in Figure 7.1b), rather than introducing a new symbol for the sufficient statistics. An alternative is to insert one or more samples from the distribution into the memory instead, but this would introduce some sampling noise.

In the subsequent prediction phase, we forward-generate from the SSM using the actions $\mathbf{a}_{\tau+1:T}$ to predict $\mathbf{s}_{\tau+1:T}$, and we use the VAE's generative model to generate the frames $\mathbf{x}_{\tau+1:T}$ given the predicted states and the information from the first $\tau$ time steps stored in the DND memory; see Section 7.3.1 for details. In our experiments, a low-dimensional state vector $\mathbf{s}_t$ (2- or 3-dimensional) suffices. Because of this, we can perform efficient rollouts in latent space without the need to generate high-dimensional frames at each time step as in autoregressive models (Oh et al., 2015; Chiappa et al., 2017; Gemici et al., 2017). Also, thanks to the scalability properties of the DND memory, we can efficiently explore very large environments.

There are three key components that define the GTM-SM and that will be introduced in the following, namely the *generative model*, the *inference network*, and the *past encoder*. As we will see, these components share many parameters.

### 7.3.1 Generative model

For brevity, we write the observations and actions in the memorization phase as $\mathbf{v} = \{\mathbf{x}_{1:\tau}, \mathbf{a}_{1:\tau}\}$ and we write the observations and actions in the prediction phase as $\mathbf{x} = \mathbf{x}_{\tau+1:T}$ and $\mathbf{a} = \mathbf{a}_{\tau+1:T}$ respectively. Letting $\theta$ be the parameters of the generative model, we model $p_\theta(\mathbf{x}|\mathbf{a}, \mathbf{v})$ as follows.

We introduce two sets of latent variables: the frame encodings $\mathbf{z} = \mathbf{z}_{\tau+1:T}$ and the SSM states $\mathbf{s} = \mathbf{s}_{\tau+1:T}$, and define the joint probability density $p_\theta(\mathbf{x}, \mathbf{z}, \mathbf{s}|\mathbf{a}, \mathbf{v})$ following the factorization shown in Figure 7.1a:

$$p_\theta(\mathbf{x}, \mathbf{z}, \mathbf{s}|\mathbf{a}, \mathbf{v}) = \prod_{t=\tau+1}^{T} p_\theta(\mathbf{x}_t|\mathbf{z}_t) p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m}) p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t), \tag{7.1}$$

where $p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t)$ is a Gaussian SSM transition probability density and $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$ is the VAE decoder (Bernoulli or Gaussian distributed, depending on the data). $p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$ can be seen as the prior of the VAE, that is conditioned at each time step on the current state $\mathbf{s}_t$ and the content of the DND memory $\mathbf{m} = \{\mathbf{s}_{1:\tau}, \mathbf{z}_{1:\tau}\} = \mathit{PastEncoder}(\mathbf{v})$ as illustrated in Figure 7.1b (see Section 7.3.3 for details on the past encoder). Its sufficient statistics are computed as follows. First, we calculate the distances $d_i = d(\mathbf{s}_i, \mathbf{s}_t)$, $i = 1, .., \tau$, between $\mathbf{s}_t$ and all the states $\mathbf{s}_i$ in the DND memory. We then retrieve from the memory the $K$ nearest states and the corresponding frame encodings, thus forming a set of triplets $\{(d^{(k)}, \mathbf{s}^{(k)}, \mathbf{z}^{(k)}), k = 1, .., K\}$ that will be used as conditioning variables when computing the parameters of the VAE prior $p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$. Using low-dimensional $\mathbf{s}_t$ and prior knowledge of the environment dynamics when defining $p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t)$, we can make the GTM-SM learn to use $\mathbf{s}_t$ to represent its position in the environment. At each time step the model will then retrieve from the memory what it has seen when it was previously close to the same location, and use this information to generate the current frame $\mathbf{x}_t$. The exact form of the VAE prior $p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$ and transition model $p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t)$ is environment-dependent, and will be therefore introduced separately for each experiment in Section 7.4.

### 7.3.2 Inference network

Due to the non-linearities in the VAE and the fact that $p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$ depends on the DND memory, the posterior distribution $p_\theta(\mathbf{z}, \mathbf{s}|\mathbf{x}, \mathbf{a}, \mathbf{v})$ of the GTM-SM is intractable. We therefore introduce a variational approximation $q_\phi(\mathbf{z}, \mathbf{s}|\mathbf{x}, \mathbf{a})$ that factorizes as

$$\begin{aligned} q_\phi(\mathbf{z}, \mathbf{s}|\mathbf{x}, \mathbf{a}) &= q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{s}|\mathbf{a}) \\ &= \prod_{t=\tau+1}^{T} q_\phi(\mathbf{z}_t|\mathbf{x}_t) p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t) \ . \end{aligned} \tag{7.2}$$

A graphical representation of the inference network of the GTM-SM is shown in Figure 7.1c. $q_\phi(\mathbf{z}_t|\mathbf{x}_t)$ is an inference network that outputs the mean and variance of a Gaussian distribution, as typically done in VAEs. In (7.2) we then use the SSM transition probabilities, and we are therefore assuming that the GTM-SM can learn the prior dynamics of the moving agents accurately enough to infer the position of the agent given the sequence of actions. Notice that without this assumption it would be impossible to perform long term generation with the model during the prediction phase. In this phase, we can in fact only rely on the generative model, and not on the inference network as we do not know what the agent is seeing at each time step. To relax this assumption, the inference network could be extended to make use of the information stored in memory, for example by using landmark information when inferring the current position of the agent. This is discussed more in detail in Appendix 7.10 in the supplementary material, together with an initial experiment to assess the feasibility of the proposed method.

### 7.3.3 Past encoder

The past encoder is used during the memorization phase to extract the information to store in the DND memory. It creates a mapping from $\mathbf{s}_t$ to $\mathbf{z}_t$ that is exploited at times $t = \tau + 1 : T$ in the generative model. During the memorization phase, at each time step we store in the DND the sufficient statistics of the inferred states $\mathbf{s}_t$ and visual information $\mathbf{z}_t$, $t = 1, .., \tau$, obtained from an approximation to the smoothed posterior $p_\theta(\mathbf{z}_t, \mathbf{s}_t|\mathbf{v})$. We first factorize this distribution as $p_\theta(\mathbf{z}_t, \mathbf{s}_t|\mathbf{v}) = p_\theta(\mathbf{s}_t|\mathbf{v})p_\theta(\mathbf{z}_t|\mathbf{v})$ and only condition on the information up to time $t$ (i.e. we are doing *filtering* instead of smoothing): $p_\theta(\mathbf{z}_t, \mathbf{s}_t|\mathbf{v}) \approx p_\theta(\mathbf{s}_t|\mathbf{v}_{1:t})p_\theta(\mathbf{z}_t|\mathbf{v}_{1:t})$, where $\mathbf{v}_{1:t} = \{\mathbf{x}_{1:t}, \mathbf{a}_{1:t}\}$ Second, we approximate each of the terms using the inference network, without introducing any additional parameters in the model. Using the VAE encoder, we have $p_\theta(\mathbf{z}_t|\mathbf{v}_{1:t}) \approx q_\phi(\mathbf{z}_t|\mathbf{x}_t)$. We then assume $p_\theta(\mathbf{s}_t|\mathbf{v}_{1:t}) \approx p_\theta(\mathbf{s}_t|\mathbf{v}_{1:t-1}, \mathbf{a}_t)$ with

$$p_\theta(\mathbf{s}_t|\mathbf{v}_{1:t-1}, \mathbf{a}_t) = \int p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t)p^\star_\theta(\mathbf{s}_{t-1})\ \mathrm{d}\mathbf{s}_{t-1}\ ,$$

$$p^\star_\theta(\mathbf{s}_t) = \int p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t)p^\star_\theta(\mathbf{s}_{t-1})\ \mathrm{d}\mathbf{s}_{t-1}\ .$$

$p^\star_\theta(\mathbf{s}_t)$ is the marginal distribution of $\mathbf{s}_t$ obtained by integrating over the past states (samples from $p^\star_\theta(\mathbf{s}_t)$, as needed in the ELBO, are easily obtained with ancestral sampling).

### 7.3.4 Training

We learn the parameters $\theta$ and $\phi$ of the GTM-SM by maximizing the ELBO, a lower bound to the log-likelihood $\log p_\theta(\mathbf{x}|\mathbf{a}, \mathbf{v})$ obtained using Jensen's inequality and the inference network introduced in Section 7.3.2:

$$\begin{aligned}\log p_\theta(\mathbf{x}|\mathbf{a}, \mathbf{v}) &= \log \int p_\theta(\mathbf{x}, \mathbf{z}, \mathbf{s}|\mathbf{a}, \mathbf{v})\ \mathrm{d}\mathbf{z}\ \mathrm{d}\mathbf{s} \\ &\geq \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{s}|\mathbf{x},\mathbf{a},\mathbf{v})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z}, \mathbf{s}|\mathbf{a}, \mathbf{v})}{q_\phi(\mathbf{z}, \mathbf{s}|\mathbf{x}, \mathbf{a}, \mathbf{v})}\right] = \mathcal{F}(\theta, \phi)\ .\end{aligned}$$

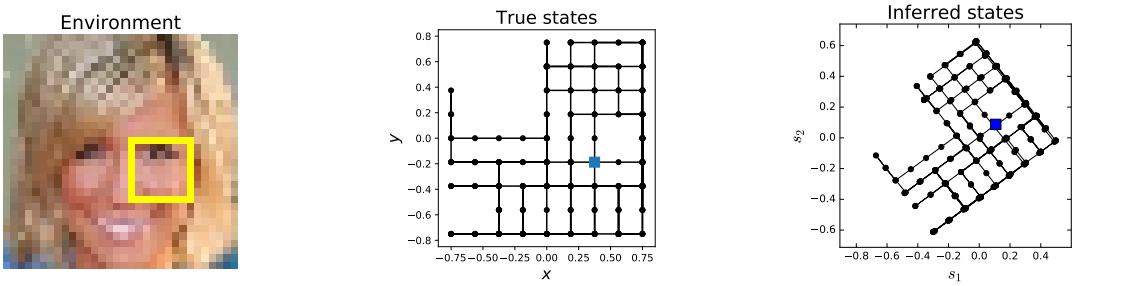
Exploiting the temporal factorization of both the joint distribution $p_\theta(\mathbf{x}, \mathbf{z}, \mathbf{s}|\mathbf{a}, \mathbf{v})$ and the variational approximation $q_\phi(\mathbf{z}, \mathbf{s}|\mathbf{x}, \mathbf{a}, \mathbf{v})$, we obtain after some calculations:

$$\mathcal{F}(\theta, \phi) = \sum_{t=\tau+1}^{T} \mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{x}_t)}\left[\log p_\theta(\mathbf{x}_t|\mathbf{z}_t)\right] - \mathbb{E}_{p^\star_\theta(\mathbf{s}_t)}\left[KL[q_\phi(\mathbf{z}_t|\mathbf{x}_t)||p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})]\right]\ .$$

The ELBO is then formed by two terms: a reconstruction term and a KL divergence for the VAE. $\mathcal{F}(\theta, \phi)$ can be maximized with stochastic gradient ascent, approximating the intractable expectations with Monte Carlo integration with a single sample and using the reparameterization trick to obtain low-variance gradients (Kingma and Welling, 2014; Rezende et al., 2014).

## 7.4 Experiments

We test the memorization and long-term generation capabilities of the GTM-SM on several 2D and 3D environments of increasing complexity. We use videos with action data from RL




*(a) Environment and current observation (the yellow square).*

*(b) Ground truth positions explored during the 256 steps of the memorization phase. The blue square represents the final position.*

*(c) Inferred positions during the 256 steps of the memorization phase, $\mathbf{s}_t = (s_1, s_2)$.*


*(d) Comparison between true and predicted frames during the prediction phase of a test sequence (time steps $t_{257:264}$ and $t_{464:471}$).*

***Figure 7.2:*** *Image navigation experiment*

agents walking in the environments, that are split so that both the memorization and prediction phase are hundreds of time steps. Experimental details can be found in Appendix 7.7 in the supplementary material.

### 7.4.1 Image navigation experiment

In this experiment the data-generating agent walks on top of an image and observes a cropped version of the image (centered at the agent's position). As illustrated in Figure 7.2a, the 2D environment is a 32x32 image from the CelebA dataset (Liu et al., 2015) and the agent sees an 8x8 crop (the yellow square in the figure). There are five possible actions: move one step up/down/left/right or stay still. At each time step we sample a random action, but to favor exploring the whole environment the action is repeated in the subsequent time steps. The number of repetitions is sampled from a Poisson distribution. The agent cannot walk outside of the image: the "move right" action, for example, does not change the position of an agent on the right edge of the image. We can interpret this as an environment with walls. The agent walks on the image while adding information in the DND memory for $\tau = 256$ time steps. We experimentally determined that this suffices to ensure that the agent usually reaches most positions in the environment. During training the prediction phase has 32 time steps; during testing we instead generate from the model for 256 time steps, so that $T = 512$. In each of the two dimensions, there are nine possible positions (the crops can overlap). This is illustrated in Figure 7.2b, which shows the ground truth positions that the agent has visited in the 256 steps of the memorization phase of a test sequence.

We use a 2-dimensional state space that the GTM-SM learns to use to represent the position of

the agent. With no walls in the environment all possible transitions are linear, and they can be modelled as $\mathbf{s}_t = \mathbf{s}_{t-1} + M\mathbf{a}_t + \boldsymbol{\varepsilon}_t$ with $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, r^2\mathbf{I})$. In all experiments we use small values for $r^2$, that make the transitions close to being deterministic. The transition matrix $M$ can be learned from the data, and describes how to update the state given each of the 5 actions ($M\mathbf{a}_t$ is the *displacement* at time $t$). The environment in our experiment has walls however, and we need the model to be able to learn not to move with actions that would make it hit a wall. We do this by multiplying the displacement by a neural network $\sigma_d$ that receives as input the projected position of the agent after taking the action and outputs a value between 0 and 1, and that can therefore learn to cancel out any displacements that would bring the agent out of the environment. These non-linear transitions are therefore modelled as

$$\mathbf{s}_t = \mathbf{s}_{t-1} + M\mathbf{a}_t \cdot \sigma_d(\mathbf{s}_{t-1} + M\mathbf{a}_t) + \boldsymbol{\varepsilon}_t \ . \tag{7.3}$$

The VAE prior used in this experiments is obtained by creating a mixture distribution from the sufficient statistics of the frame encodings retrieved from the DND memory, whose weights are inversely proportional to the squared distances $d^{(k)}$ between $\mathbf{s}_t$ and the retrieved elements $\mathbf{s}_k^{(k)}$:

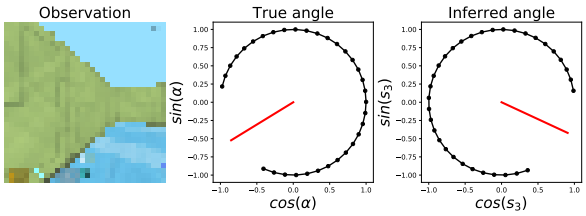
$$p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m}) = \sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{z}_t|\mathbf{z}^{(k)}) \ ; \quad w_k \propto \frac{1}{{d^{(k)}}^2 + \delta}$$

$\delta = 10^{-4}$ is added for numerical stability (Pritzel et al., 2017). In the DND memory we store sufficient statistics, but in this experiment we use the Euclidean distance between means in the nearest-neighbor search. (Alternatively, we could use the KL divergence between the distributions).

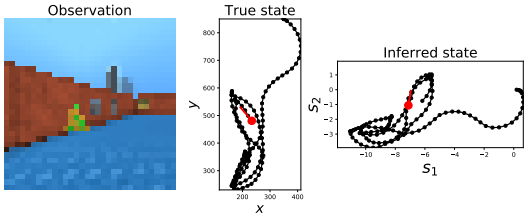
In Figure 7.2c we show an example of the states inferred by the model for a test sequence. We see that the model has learned the correct transitions, in a state space that is rotated and stretched with respect to the ground truth one. To test the memorization and prediction capabilities of the GTM-SM, Figure 7.2d shows a comparison between the ground truth frames of the video and the predicted ones during the prediction phase. The model produces almost perfect predictions, even after more than 200 generation steps ($t = 471$). This shows that it has learned to store all relevant information in the DND, as well as retrieve all relevant information from it. Videos of long-term generations from the model are available in the supplementary material, see Appendix 7.8 for details. The state-of-the-art generative temporal models with memory introduced in (Gemici et al., 2017), are not able to capture the spatial structure of large environments as in the GTM-SM, and would therefore struggle to coherently generate hundreds of time steps into the future. The MNIST maze experiment in (Gemici et al., 2017) can be seen as a simpler version of the image navigation experiment presented above, with agents moving on a 4x4 grid, linear transitions and 25-step sequences.

### 7.4.2 Labyrinth experiments

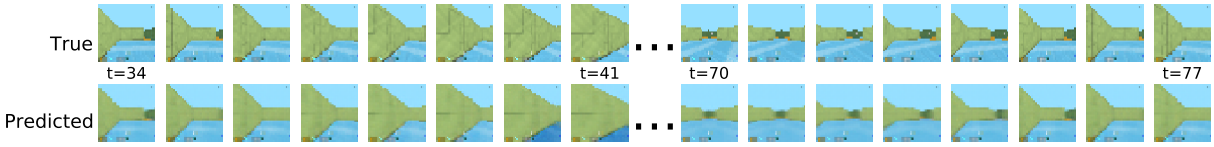
We now show that the GTM-SM is able to remember the past and perform spatio-temporally coherent generations over hundreds of time steps in simulated 3D environments. We use the *Labyrinth* environment (Mnih et al., 2016; Beattie et al., 2016), procedurally-generated 3D mazes with random textures, objects and wall configurations. There are eight possible actions that can both move and rotate the agent in the maze, and we observe images from a first-person point of view. Labyrinth can be seen as a 3D extension of the image navigation experiments in Section



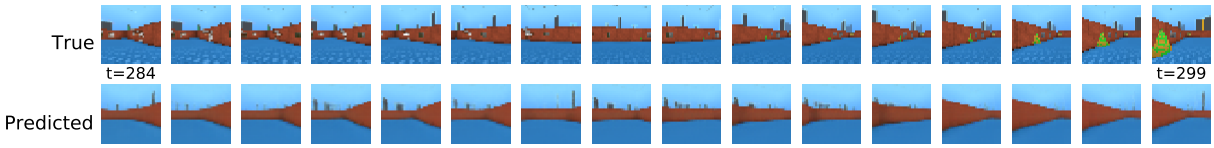

*(a)* ***Rotating agent****, environment and angle information for $t = 38$. The red line denotes the current orientation of the agent, the black points indicate the orientations explored in the memorization phase. $\alpha$ denotes the ground truth orientation.*




*(b)* ***Walking agent****, environment and state information for $t = 297$. The red circle and line represent the current position and orientation of the agent respectively, the black points indicate the states explored in the memorization phase.*




*(c)* ***Rotating agent*** *in Labyrinth, prediction phase of a test sequence. Time steps $t_{34:41}$ and $t_{70:77}$.*




*(d)* ***Walking agent*** *in Labyrinth, prediction phase of a test sequence. Time steps $t_{284:299}$.*

***Figure 7.3:*** *Labyrinth experiment*

7.4.1, but in this case the task is much harder for two main reasons that will be tackled below: (1) dealing with rotations and (2) projective transformations in a partially observable environment.

First, the state of the agent is no longer only described by the position, but also from the direction in which the agent is looking and moving. We need to take into account two different coordinate systems, a global one that coincides with the state-space ($\mathbf{s}$-space), and one that is fixed with the agent (agent-space). In the image navigation experiments these coordinate systems coincided. The actions act in agent-space, e.g. a "move right" action will make the agent go right in its reference frame, but depending on its orientation this could correspond to a move to the left in $\mathbf{s}$-space. To deal with this issues we can introduce a *rotation matrix* $R$ in the state transition equation, that translates a displacement $M\mathbf{a}_t$ in agent-space to a displacements $RM\mathbf{a}_t$ in $\mathbf{s}$-space. More in detail, we consider a 3-dimensional state-space, and define the state transition equations as

$$\mathbf{s}_t = \mathbf{s}_{t-1} + R(\mathbf{s}^{(3)}_{t-1})M\mathbf{a}_t + \boldsymbol{\varepsilon}_t \tag{7.4}$$

with $\mathbf{s}^{(3)}_{t-1}$ being the 3rd component of the vector $\mathbf{s}_{t-1}$ and

$$R(\mathbf{s}^{(3)}_{t-1}) = \begin{bmatrix} \cos(\mathbf{s}^{(3)}_{t-1}) & -\sin(\mathbf{s}^{(3)}_{t-1}) & 0 \\ \sin(\mathbf{s}^{(3)}_{t-1}) & \cos(\mathbf{s}^{(3)}_{t-1}) & 0 \\ 0 & 0 & 1 \end{bmatrix} .$$

As for the image navigation experiment, we learn the parameters of $M$. While we do not explicitly tell the GTM-SM to use the first two component of the state vector as a position and the third

one as an angle, the model will learn to use them in this way in order to maximize the ELBO. In the nearest neighbor search in the DND memory we need to take into account the periodicity of the angle, e.g. that an agent oriented at 30° or 390° is actually looking in the same direction. When computing distances, instead of using $\mathbf{s}_t^{(3)}$ we then use $\cos(\mathbf{s}_t^{(3)})$ and $\sin(\mathbf{s}_t^{(3)})$, that are possibly passed together with the first two components of $\mathbf{s}_t$ through a linear layer that maps the resulting vector in a learned space where we use the Euclidean distance.

The second challenge arises from the fact that, unlike the image navigation experiment where there were a limited a number of possible positions for the agent, in the 3D labyrinth environment it is not reasonable to assume that during the memorization phase the agent will pass in all positions and look from them in all directions. To deal with this issue, we use as VAE prior $p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$ a neural architecture that given the frames from the closest positions retrieved from the DND memory learns to combine the different views taking into account projective transformations. Details can be found in Appendix 7.7.2. Videos of long-term generations for these experiments are available in the supplementary material, see Appendix 7.8 for details.

#### 7.4.2.1 Rotating agent in Labyrinth

In the first experiment we test the abilities of the GTM-SM to learn to model rotations with the transition model in (7.4) as well as to combine the information from different views. We use videos with action data of an agent that does two complete rotations while standing still in the same position. The rotational period is around 41 time steps, but we only store in memory the first $\tau = 33$ (approximately 300°). We then ask the model to generate the remaining 60° to finish the first rotation and a whole new rotation. From Figure 7.3a we see an example of an observation from the test data and that the model has correctly learned to use the third component of the state vector $\mathbf{s}_t$ to represent the orientation of the agent. In the prediction phase in Figure 7.3c, we notice that the predictions from the model are very close to the ground truth, meaning that the model has learned to use the memory correctly. In particular, despite the fact that the frames from $t = 33$ to $t = 41$ were never seen during the memorization phase, the GTM-SM has learned to combine the information from other views. Notice that this experiment can be seen as a more challenging version of the Labyrinth rotation experiment of (Gemici et al., 2017), that used a fully observed first rotation with a rotational period of 15 time steps.

#### 7.4.2.2 Walking agent in Labyrinth

We now use videos of a pre-trained RL agent walking in a room and solving a scavenger hunt task. In this case it is fundamental to extend the transition equation in (7.4) to model more carefully the physics of the walking agent, that make the displacement at a given time step depend not only on the current action, but also on the displacement at the previous time step. The agent is subject to momentum and friction, so that if it is moving in a certain direction at time $t-1$, it will still continue to move a bit in the same direction even at time $t$, regardless of the action $\mathbf{a}_t$. Also, despite the momentum, the displacement of the agent cannot increase indefinitely, i.e. there is saturation. We can model this by extending the way the displacement $\mathbf{d}_t = M\mathbf{a}_t$ is calculated in (7.4). To take into account momentum and friction, we first add to $\mathbf{d}_t = M\mathbf{a}_t$ a damped version of the displacement at the previous time step, i.e. $\sigma(\mathbf{c}_f) \odot \mathbf{d}_{t-1}$, where $\sigma(\mathbf{c}_f)$ is a learned

vector between 0 and 1 of the same size of $\mathbf{d}_t$ ($\sigma$ is the sigmoid function and $\odot$ represents the element-wise product). To deal with saturation, we then limit the range of the displacements by squashing them through a tanh non-linearity that is pre-multiplied by a learned vector $\mathbf{c}_s$. The resulting transition model then becomes

$$\mathbf{s}_t = \mathbf{s}_{t-1} + R(\mathbf{s}^{(3)}_{t-1}) \underbrace{\mathbf{c}_s \odot \tanh(\sigma(\mathbf{c}_f) \odot \mathbf{d}_{t-1} + M\mathbf{a}_t)}_{\mathbf{d}_t} + \boldsymbol{\varepsilon}_t$$

with $M$, $\mathbf{c}_s$ and $\mathbf{c}_f$ parameters to be learned. This is a very challenging task, as from only one-hot encoded actions and the frames of the video we need to learn a complex transition model. Moreover, due to the low resolution of the images (32x32), small variations in state-space may be impossible to infer using only the images. To solve this, we make the reasonable assumption that at training time the agent feels its movement, i.e. the displacements. We then add a regression loss as an extra term to the objective function, that helps the GTM-SM to learn transition parameters such that the estimated $\mathbf{d}_t$ is close to its true value. Notice that the true displacements information is only added as a target in the loss, and never passed directly into the model. The pre-trained agent does not hit walls, therefore we do not need to handle non-linearities as in (7.3).

We let the agent walk around the room while adding information in the DND memory for $\tau = 150$ time steps, and then predict during testing the following 150 time steps ($T = 300$). In Figure 7.3b, we notice that the GTM-SM is able to learn a very accurate transition model, that provides a sufficiently good approximation of the true state even after $t = 297$ time steps. In Figure 7.3d we can appreciate the memorization and long-term generation capabilities of the GTM-SM by looking at the comparison between the true and predicted frames of the video in the end of the prediction phase ($t_{284:299}$). We also notice in the predicted frames, that the model correctly draws the walls and the floors but fails to render the objects, probably due to difficulties in modelling with the VAE the very diverse and complex textures that form objects in this environment. We also tested the same trained model on longer videos of larger environments with multiple rooms ($\tau = 150$ and $T = 450$). As explained in detail in Appendix 7.9, the model is able to correctly predict the textures of the environment even after 300 time steps.

## 7.5 Related work

A number of recent works have augmented deep generative models with learned external memories, both in the static setting (Li et al., 2016; Bornschein et al., 2017) and in the temporal one (Gemici et al., 2017). More in general, neural networks have been combined with different memories in a wide range of tasks such as supervised learning (Graves et al., 2014; Graves et al., 2016), reinforcement learning (Oh et al., 2016; Pritzel et al., 2017; Parisotto and Salakhutdinov, 2018), one-shot learning (Santoro et al., 2016), question answering and language modelling (Sukhbaatar et al., 2015; Miller et al., 2016). Each memory architecture uses different addressing mechanisms to write or read information, that are usually chosen depending on the specific application being considered. As discussed in the introduction, our work is closely related to (Gemici et al., 2017), but more suitable for long-term generation for this task and more scalable thanks to the usage of the spatial memory architecture that exploits knowledge on the dynamics of the agent and does not require to learn a parametric memory addressing scheme.

In the deep reinforcement learning community, several works have exploited different memory architectures to store long term information to be used within an agent's policy, such as in (Zaremba and Sutskever, 2015; Oh et al., 2016). In particular, in (Gupta et al., 2017a; Gupta et al., 2017b; Zhang et al., 2017; Parisotto and Salakhutdinov, 2018) the memory architectures have a fixed number of slots that are spatially structured as a 2D grid, and can therefore store information on the moving agent. Similarly to the GTM-SM, these memories are built to exploit the spatial structure of the problem, although for the different task of constructing agents that can learn to navigate and explore the environment, as opposed to the focus on generative modelling of this paper. Simultaneous Localization And Mapping (SLAM) (Smith et al., 1987; Leonard and Durrant-Whyte, 1991) is a popular technique used in robotics to estimate the position of a robot and the map of the environment at the same time using sensor data, recently applied to deep reinforcement learning for example in (Bhatti et al., 2016; Zhang et al., 2017). It is reminiscent to the memorization phase of the GTM-SM, that could be therefore extended using ideas introduced in the visual SLAM community (Taketomi et al., 2017).

## 7.6 Conclusion

In this work we introduced the Generative Temporal Model with Spatial Memory, an action-conditioned generative model that uses a scalable non-parametric memory to store spatial and visual information. Our experiments on simulated 2D and 3D environments show that the model is able to coherently memorize and perform long-term generation. To our knowledge this is the first published work that builds a generative model for agents walking in an environment that, thanks to the separation of the dynamics and visual information in the DND memory, can coherently generate for hundreds of time steps in a scalable way. Future work will focus on exploiting these capabilities in model-based planning by integrating the GTM-SM within an RL agent.

# Supplementary material

## 7.7 Experimental details

The models used in all experiments are implemented in Tensorflow (Abadi et al., 2015) and use the Adam optimizer (Kingma and Ba, 2014).

### 7.7.1 Image navigation experiment

The training and test data sets are procedurally generated by sampling a random trajectory in randomly chosen images from the CelebA data set. The actions at each time steps are one-hot encoded (vector of size 5). The memorization phase is 256 time steps, while the prediction one has 32 time steps during training and 256 during testing.

The VAE decoder and encoder use a 3-layered convolutional architecture to parameterize mean and variance of 16-dimensional latent states, but we noticed in practice that for this experiment even standard fully connected architectures perform well. In the transition model the standard deviation of the model is $r = 10^{-3}$. In the DND we retrieve the 5 nearest neighbour and use Euclidean distances between means.

The initial learning rate is $10^{-3}$, and we anneal it linearly to $5 \cdot 10^{-5}$ during the first 50000 updates.

### 7.7.2 Labyrinth experiments

The data sets used for the labyrinth experiments contain 120000 action-conditioned videos, of which we use 100000 for training and 20000 for testing. Each video for the rotating agent experiments contains 80 frames. To form a training sequence we select randomly 49 consecutive frames of a video, that we split in 33 frames for the memorization phase and 16 for the prediction one. During testing, the prediction phase has 45 time steps. For the walking agent experiment the videos are 300 time steps. Similarly to the rotation experiment, to form a training sequence we get consecutive sequences of 150+32 time steps (memorization and prediction phase respectively). During testing, we use 150 frames for the prediction phase.

The transition noise of the SSM has standard deviation $r = 10^{-2}$. To compute distances in the DND we map the state vectors to

$$\widetilde{\mathbf{s}}_t = \begin{bmatrix} \mathbf{s}_t^{(1)} \\ \mathbf{s}_t^{(2)} \\ \cos(\mathbf{s}_t^{(3)}) \\ \sin(\mathbf{s}_t^{(3)}) \end{bmatrix},$$

and optionally pass the resulting vector through a linear layer. This gives a 5-dimensional vector in a learned manifold in which we use the Euclidean distance (in our experiments, the model performed well even without the linear layer). In the DND we retrieve the 4 nearest neighbours.

In the VAE, we use convolutional encoder and decoder, and 64-dimensional latent state. The VAE prior $p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$ used in the Labyrinth experiments is slightly more involved than the mixture prior used in the image navigation one. We first map the data retrieved from memory $\{\mathbf{s}_i, \mathbf{z}_i\}$ with a MLP to an embedding vector $h_t$, and then combine the embedding $h_t$ with the current state $\mathbf{s}_t$, mapping the result to the mean and variance of the Gaussian prior

$$\begin{aligned} h_t &= f(\{\mathbf{s}_i, \mathbf{z}_i\}) \\ p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m}) &= \mathcal{N}(\boldsymbol{\mu}(h_t, \mathbf{s}_t), \boldsymbol{\sigma}(h_t, \mathbf{s}_t)). \end{aligned}$$

The initial learning rate is set to $3 \cdot 10^{-3}$, and we linearly anneal it to $5 \cdot 10^{-5}$ during the first 100000 updates.

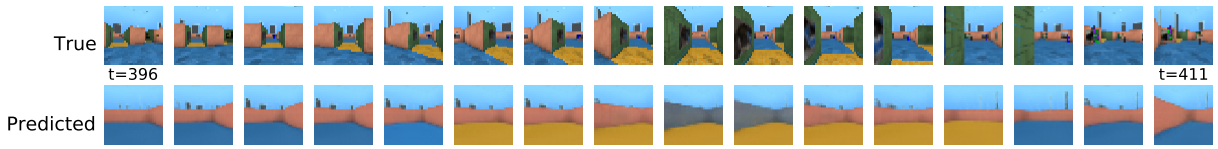



***Figure 7.4:*** *Prediction phase for a walking agent trained on one room and tested on multiple rooms. Time steps* $t_{396:411}$.

## 7.8 Videos of long-term generation

Videos of long-term generation from the GTM-SM for all the experiments of this paper are available at this anonymous Google Drive link (goo.gl/RXQPTL). The videos are subdivided in folders:

1. `videos/image_navigation/` contains the videos for the experiments in Section 7.4.1.
2. `videos/labyrinth_rotation/` contains the videos for the experiments in Section 7.4.2.1.
3. `videos/labyrinth_walk/` contains the videos for the experiments in Section 7.4.2.2.
4. `videos/labyrinth_walk_multirooms/` contains the videos for the experiments in Appendix 7.9.

In all folders, the first video corresponds to the test sequence used to produce the figures in the paper.

## 7.9 Walking agent in Labyrinth (multiple rooms)

We consider the same trained model used for the results in section 7.4.2.2. In section 7.4.2.2, this model was tested on videos of length $T = 300$ of an agent walking in a single room, with both memorization and prediction phase of 150 time steps. To asses the long-term memorization and localization capabilities of the GTM-SM, we now test it on videos of the same agent walking in larger environments with multiple rooms. Each video is $T = 450$ time steps; we store 150 time steps in memory and we predict for 300 more. As the model is trained on single rooms, we cannot expect the VAE to correctly generate the corridors between rooms, but we can expect the model to be able to know its position and the textures in the room (i.e. the color of the walls and of the floor).

In Figure 7.4 we show the predictions from the model after more than 250 time steps from the end of the memorization phase. As expected, the model fails in drawing the walls that form the corridor between the two rooms. However, we see that the GTM-SM correctly remembers the texture of rooms that it has previously visited and is able to predict the change in the color of the floor in the corridor. This is better viewed looking at the videos of this experiment, available in the folder `videos/labyrinth_walk_multirooms/` in the supplementary material.

## 7.10 Inference network using landmark information

We now introduce an alternative inference network that uses the information in the DND memory to improve inference in cases in which the SSM transition model is not powerful enough to infer the correct position of the agent. We factorize the variational approximation $q_\phi(\mathbf{z}, \mathbf{s}|\mathbf{x}, \mathbf{a}, \mathbf{v})$ as

$$\begin{aligned} q_\phi(\mathbf{z}, \mathbf{s}|\mathbf{x}, \mathbf{a}, \mathbf{v}) &= q_\phi(\mathbf{z}|\mathbf{x}) q_\phi(\mathbf{s}|\mathbf{z}, \mathbf{a}, \mathbf{v}) \\ &= \prod_{t=\tau+1}^{T} q_\phi(\mathbf{z}_t|\mathbf{x}_t) q_\phi(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t, \mathbf{z}_t, \mathbf{m}) \ . \end{aligned}$$

A graphical representation of the inference network of the GTM-SM is shown in Figure 7.5. $q_\phi(\mathbf{z}_t|\mathbf{x}_t)$ is an inference network that outputs the mean and variance of a Gaussian distribution, as typically done in VAEs. The structured variational approximation $q_\phi(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t, \mathbf{z}_t, \mathbf{m})$ retains the temporal dependency among the state variables and exploits the information stored in the memory $\mathbf{m}$. We define this approximation to depend on:

1. The *prior belief* $p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t)$. If at time $t-1$ we were at a given position, this dependence captures the fact that at time $t$ we cannot be too far from it. This is the same dependence we used in Section 7.3.2.

2. *Landmark information*, obtained by querying the DND memory in the reverse direction with respect to the VAE prior, i.e., considering the frame encodings $\mathbf{z}_i$ as keys and the states $\mathbf{s}_i$ as values. At each time step the agent can then check whether it has already seen the current frame encoding $\mathbf{z}_t$ in the past, and exploit this information when computing the inferred position of the agent. We use $\mathbf{z}_t$ to query the reversed-DND, retrieving triplets $\{(\delta^{(k)}, \mathbf{z}^{(k)}, \mathbf{s}^{(k)}), k = 1, .., K'\}$ that are used in the computation of the parameters of $q_\phi(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t, \mathbf{z}_t, \mathbf{m})$. Here, $\delta_i^{(k)}$ represents a distance in $\mathbf{z}$-space.

We define $q_\phi(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t, \mathbf{z}_t, \mathbf{m})$ to be a Gaussian density, whose mean $\boldsymbol{\mu}_q$ and variance $\boldsymbol{\sigma}_q^2$ are the outputs of a neural network that merges the sufficient statistics $\boldsymbol{\mu}_p$ and $\boldsymbol{\sigma}_p^2$ of the prior $p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_t)$, and the ones of the states $\mathbf{s}_i^{(k)}$ retrieved using landmark information: $\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2, k = 1 : K'$. We assume that we stored in the DND the mean and the variance of Gaussian latent states. The posterior mean is obtained as

$$\boldsymbol{\mu}_q = \boldsymbol{\mu}_p + \sum_{k=1}^{K'} \beta_k (\boldsymbol{\mu}_k - \boldsymbol{\mu}_p) \ ,$$

where $\beta_k \in [0, 1]$ is the output of a simple neural network with input $\delta_i^{(k)}$. The inference network can then learn to assign a high value to $\beta_k$ whenever the distance in $\mathbf{z}$-space is small (i.e. the current observation is similar to a frame stored in the DND), so that the prior mean is moved in the direction of $\boldsymbol{\mu}_k$. Similarly, the posterior variance can be computed starting from the prior variance using another neural network: $\log \boldsymbol{\sigma}_q^2 = \log \boldsymbol{\sigma}_p^2 + NN(\delta_i^{1:K'}, \boldsymbol{\sigma}_{1:K'}^2, \boldsymbol{\sigma}_p^2)$.

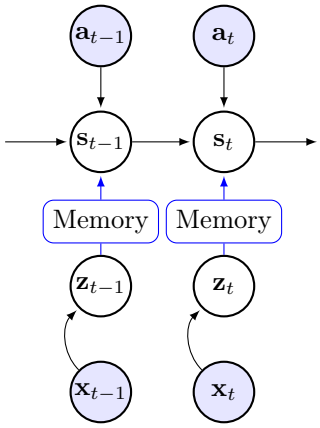



***Figure 7.5:*** *Inference network for the GTM-SM using landmark information. Blue arrows represent dependencies on the reversed DND memory.*

With this choice for the inference network, the ELBO of the GTM-SM becomes

$$\mathcal{F}(\theta,\phi) = \sum_{t=\tau+1}^{T} \mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{x}_t)}\left[\log p_\theta(\mathbf{x}_t|\mathbf{z}_t)\right] - \mathbb{E}_{q^\star_\phi(\mathbf{s}_t)}\left[KL[q_\phi(\mathbf{z}_t|\mathbf{x}_t)||p_\theta(\mathbf{z}_t|\mathbf{s}_t,\mathbf{m})]\right] + \\ - \mathbb{E}_{q^\star_\phi(\mathbf{s}_{t-1})}\left[KL\left[q_\phi(\mathbf{s}_t|\mathbf{s}_{t-1},\mathbf{a}_t,\mathbf{z}_t,\mathbf{m})||p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1},\mathbf{a}_t)\right]\right] \ .$$

with

$$q^\star_\phi(\mathbf{s}_t) = \int q_\phi(\mathbf{s}_t|\mathbf{s}_{t-1},\mathbf{a}_t,\mathbf{z}_t,\mathbf{m}_{1:t-1}) q^\star_\phi(\mathbf{s}_{t-1}) \ \mathrm{d}\mathbf{s}_{t-1} \ .$$

Notice in particular the additional KL term for the SSM.

### 7.10.1 Image navigation with obstacles

We extend the image navigation experiments of Section 7.4.1 adding obstacles to the environment as illustrated in Figure 7.6 (left). We use displacement information as in the Labyrinth experiment of Section 7.4.2.2. The obstacles appear in random positions in each sequence, therefore we cannot learn a prior transition model that captures these non-linear dynamics. However, when doing inference the model can use its knowledge on the current frame $\mathbf{x}_t$ (that is not available during the prediction phase) to infer its position by exploiting landmark information.

To illustrate this we can look at the example in Figure 7.6 (right). At time $t-1$, the position $\mathbf{s}_{t-1}$ of the agent coincides with the red star. The agent's position together with the corresponding observation $\mathbf{x}_{t-1}$ (the yellow square) will be inserted in the DND memory. At time $t$, the agent receives a "move left" action; the prior transition probabilities will then predict that the agent has to move to the left (the green hexagon). Due to the presence of the obstacle however, the agent does not move, meaning that $\mathbf{x}_t$ will be the same as $\mathbf{x}_{t-1}$. Querying the DND in the reverse direction the model will then know that the inferred state (the blue dot) should be the same as the position at the previous time step that was stored in the DND (the red star).

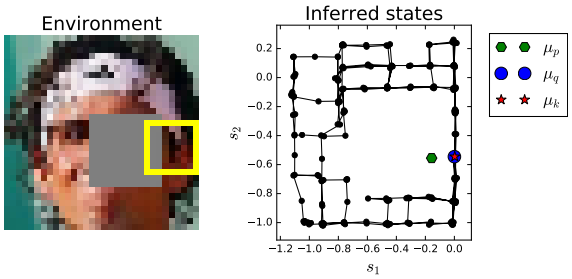



***Figure 7.6:*** *Image navigation experiment with obstacles. The agent (yellow square) cannot cross the ostacle (the gray squared area).*

# Part III

# Closing

Chapter 8

# Conclusions

## 8.1 Contributions

The ever-increasing availability of unlabelled sequential data is driving the development of mathematical models and methods that can be used to analyze it in a general and scalable way. This thesis represents a step in this direction, inspired by the recent successes in probabilistic modelling and deep learning.

In the first part of the thesis we developed a unified framework that merges ideas from latent variable models, state-space models and deep learning, and defined a broad class of deep latent variable models for sequential data that are:

1. *Flexible.* These models can fit complex data distributions in a wide range of applications. We achieve this by constructing probabilistic sequential models in which we use deep neural networks to parameterize the conditional distributions that define them. Deep learning architectures form very expressive function approximators, and allow the model to learn to perform an automatic feature extraction that is fundamental for the models to be broadly applicable. Also, new advancements in deep learning can be easily incorporated in this framework.

2. *Scalable.* Since both the generative model and the posterior approximation are defined with deep neural networks, we can perform end-to-end training using gradient-based optimization of the ELBO, computing the required gradients efficiently with GPU implementations of the back-propagation algorithm. The usage of mini-batches and amortized inference techniques allows us to train these probabilistic models in a scalable way on very large data sets.

3. *Easy to implement.* These models can be implemented using existing deep learning libraries, that use automatic differentiations techniques to automatically compute the gradients of any differentiable architecture. Thanks to the usage of inference networks, probabilistic inference

can be seen as a simple black-box operation, as opposed to the model-specific calculations required by other approximate inference techniques such as mean-field variational inference or MCMC methods.

In the second part of the thesis we further presented three papers that introduce in depth novel models belonging to this framework. In (Fraccaro et al., 2016c, Chapter 5) we showed how we can combine the power of RNNs in capturing long-term dependencies in the data with the ability of DSSMs to model the uncertainty in the learned latent representation. (Fraccaro et al., 2017, Chapter 6) then discusses how incorporating structure to the model we can learn disentangled and more interpretable visual and dynamics representations. Finally, in (Fraccaro et al., 2018, Chapter 7) we discussed the usage of an external memory architecture to deal with applications that require a high memory capacity.

## 8.2 Open questions and future work

It is well known by researchers that every new advancement in a field comes with a new set of open questions that are often harder to answer than the initial ones. For the sequential deep latent variable models introduced in this thesis there are a number of open questions whose answer would allow us to achieve a better understanding of their working principles as well as to better exploit their modelling power. We divide them in two groups: *modelling* and *inference and learning*.

**Modelling:**

- The selection of the best model parameterization to use for a given application can be difficult. This class of models in fact inherits from its deep learning component all the challenges related to the specification of the exact network architecture, e.g. number of layers, units and the type of activation function to use. In particular, the choice of the specific parameterization of the transition/emission distributions has a large impact in terms of performances. It is then fundamental to verify if there are better default choices that consistently perform better than the ones presented in this thesis.

- As discussed in Section 4.8, if we are not careful enough during training often the model tends to set all the variances to a very low value, making the model basically deterministic. Instead of solving this issue at training time, we may be able to define architectures that can natively better exploit the stochasticity.

- It is often difficult to understand if for a particular application it is important to model stochasticity in the latent states, e.g. whether one should use an RNN or an SRNN. This is part of an even broader question on which is the exact role played by the stochastic component in such architectures. Is it really modelling uncertainty or is it just introducing some "tailored noise" that is beneficial at training time?

- More work is needed to fully understand whether the uncertainties learned by these models are meaningful and well calibrated.

**Inference and learning:**

- In these models we use inference networks to parameterize Gaussian variational approximations, and inherit therefore the same issues and possible solutions discussed for VAEs in Section 2.5. More research is needed in this direction to define even better objective functions and scalable methods to build more flexible posterior approximations.
- In this thesis we have only focused on variational methods, as they provided a scalable way to perform approximate inference. A possible line of research is to focus on more efficient ways to perform other approximate inference methods (such as MCMC), similarly to the usage of the particle filters with learned importance distribution discussed in Section 4.3.2. In some cases it is in fact desirable to be able to use more computationally expensive inference methods in order to achieve better performances.
- As discussed in Section 4.6.2, probabilistic graphical models provide a principled way to introduce prior knowledge and structure in the model, and we can leverage existing message-passing algorithms to perform approximate inference. A fundamental but challenging step for the more widespread usage of such techniques is the implementation of a message-passing library that integrates well with existing deep learning libraries.
- As always when using deep learning, training tricks can make a huge difference in terms of final performances of the model. In Section 4.8 we have introduced some of them, but it would be interesting to find even better ones derived from a deeper theoretical understanding of the learning process.

The application of these model to even more complex and challenging tasks can drive the research on many of the open questions presented above. Of particular interest is the usage of this class of models in applications for which we know that a correct model for the uncertainty is fundamental. This is the case for example in model-based reinforcement learning, where these models could be used within an agent for planning and reasoning under uncertainty.

## References


Abadi M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*.

Agakov F. and D. Barber (2004). "An Auxiliary Variational Method". In: *Neural Information Processing*. Springer Berlin Heidelberg.

Archer E., I. M. Park, L. Buesing, J. Cunningham, and L. Paninski (2015). "Black box variational inference for state space models". In: *arXiv:1511.07367*.

Barber D., A. T. Cemgil, and S. Chiappa (2011). "Inference and estimation in probabilistic time series models". In: *Bayesian Time Series Models*.

Barber D. and S. Chiappa (2007). "Unified Inference for Variational Bayesian Linear Gaussian State-Space Models". In: *Advances in Neural Information Processing Systems 19*.

Bastien F., P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio (2012). "Theano: new features and speed improvements". In: *arXiv:1211.5590*.

Battaglia P. W., R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu (2016). "Interaction Networks for Learning about Objects, Relations and Physics". In: *NIPS*.

Bayer J. and C. Osendorfer (2014). "Learning stochastic recurrent networks". In: *arXiv:1411.7610*.

Beal M. J. (2003). "Variational Algorithms for Approximate Bayesian Inference". PhD thesis. Gatsby Computational Neuroscience Unit, University College London.

Beattie C., J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen (2016). "DeepMind Lab". In: *CoRR* abs/1612.03801.

Bentley J. (1975). "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM*.

Bhatti S., A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. S. Torr (2016). "Playing Doom with SLAM-Augmented Deep Reinforcement Learning". In: *arXiv:1612.00380*.

Bishop C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 0387310738.

Bornschein J., A. Mnih, D. Zoran, and D. Jimenez Rezende (2017). "Variational Memory Addressing in Generative Models". In: *Advances in Neural Information Processing Systems 30*.

Boulanger-Lewandowski N., Y. Bengio, and P. Vincent (2012). "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription". In: *arXiv:1206.6392*.

Bowman S. R., L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio (2015). "Generating Sentences from a Continuous Space". In: *arXiv:1511.06349*.

Burda Y., R. Grosse, and R. Salakhutdinov (2015). "Importance Weighted Autoencoders". In: *arXiv preprint arXiv:1509.00519*.

Chang M. B., T. Ullman, A. Torralba, and J. B. Tenenbaum (2017). "A Compositional Object-Based Approach to Learning Physical Dynamics". In: *ICLR*.

Chen X., D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel (2017). "Variational Lossy Autoencoder". In: *International Conference on Learning Representations*.

Chiappa S., S. Racanière, D. Wierstra, and S. Mohamed (2017). "Recurrent Environment Simulators". In: *ICLR*.

Cho K., B. Van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *EMNLP*, pp. 1724–1734.

Chung J., C. Gulcehre, K. Cho, and Y. Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv:1412.3555*.

Chung J., K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio (2015). "A recurrent latent variable model for sequential data". In: *NIPS*, pp. 2962–2970.

Cremer C., X. Li, and D. Duvenaud (2018). "Inference Suboptimality in Variational Autoencoders". In: *arXiv:1801.03558*.

Deisenroth M. P. and C. E. Rasmussen (2011). "PILCO: A Model-based and Data-efficient Approach to Policy Search". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11.

Del Moral P., A. Doucet, and A. Jasra (2006). "Sequential Monte Carlo samplers". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.3, pp. 411–436.

Dempster A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistiical Society*.

Deng Z., R. Navarathna, P. Carr, S. Mandt, Y. Yue, I. Matthews, and G. Mori (2017). "Factorized Variational Autoencoders for Modeling Audience Reactions to Movies". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dieleman S., J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, E. Battenberg, and A. van den Oord (2015). *Lasagne: First release*.

Doucet A., N. de Freitas, and N. Gordon (2001). "An Introduction to Sequential Monte Carlo Methods". In: *Sequential Monte Carlo Methods in Practice*. Statistics for Engineering and Information Science, pp. 3–14.

Doucet A. and A. Johansen (2008). *A Tutorial on Particle Filtering and Smoothing: Fifteen years Later*. Tech. rep.

Doucet A., S. Godsill, and C. Andrieu (2000). "On Sequential Monte Carlo Sampling Methods for Bayesian Filtering". In: *Statistics and Computing*.

Durbin J. and S. Koopman (2012). *Time Series Analysis by State Space Methods: Second Edition*. Oxford Statistical Science Series. OUP Oxford.

Fabius O. and J. R. van Amersfoort (2014). "Variational Recurrent Auto-Encoders". In: *arXiv:1412.6581*.

Finn C., I. J. Goodfellow, and S. Levine (2016). "Unsupervised Learning for Physical Interaction through Video Prediction." In: *NIPS*.

Finn C. and S. Levine (2017). "Deep visual foresight for planning robot motion". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, pp. 2786–2793.

Fraccaro M., S. Kamronn, U. Paquet, and O. Winther (2017). "A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning". In: *Advances in Neural Information Processing Systems 30, NIPS*.

Fraccaro M., U. Paquet, and O. Winther (2016a). " An Adaptive Resample-Move Algorithm for Estimating Normalizing Constants". In: *arXiv preprint arXiv:1604.01972*.

Fraccaro M., U. Paquet, and O. Winther (2016b). *Indexable Probabilistic Matrix Factorization for Maximum Inner Product Search*.

Fraccaro M., D. Rezende, Y. Zwols, A. Pritzel, S. M. A. Eslami, and F. Viola (2018). "Generative Temporal Models with Spatial Memory for Partially Observed Environments". In: *Proceedings of the 35th International Conference on Machine Learning, ICML*. Proceedings of Machine Learning Research.

Fraccaro M., S. K. Sønderby, U. Paquet, and O. Winther (2016c). "Sequential Neural Models with Stochastic Layers". In: *NIPS*.

Fragkiadaki K., P. Agrawal, S. Levine, and J. Malik (2016). "Learning Visual Predictive Models of Physics for Playing Billiards". In: *ICLR*.

Gales M. and S. Young (2008). *The Application of Hidden Markov Models in Speech Recognition*. Foundations and trends in signal processing. Now Publishers.

Gan Z., C. Li, R. Henao, D. E. Carlson, and L. Carin (2015). "Deep temporal sigmoid belief networks for sequence modeling". In: *NIPS*, pp. 2458–2466.

Gao Y., E. W. Archer, L. Paninski, and J. P. Cunningham (2016). "Linear dynamical neural population models through nonlinear embeddings". In: *NIPS*.

Geiger D., T. Verma, and J. Pearl (1990). "Identifying independence in Bayesian Networks". In: *Networks* 20, pp. 507–534.

Gemici M., C. Hung, A. Santoro, G. Wayne, S. Mohamed, D. J. Rezende, D. Amos, and T. P. Lillicrap (2017). "Generative Temporal Models with Memory". In: *arXiv:1702.04649*.

Gershman S. J. and N. D. Goodman (2014). "Amortized Inference in Probabilistic Reasoning". In: *Proceedings of the 36th Annual Conference of the Cognitive Science Society (CogSci 2014)* 1, pp. 517–522.

Ghahramani Z. and G. E. Hinton (1996). "Parameter Estimation for Linear Dynamical Systems". In: *University of Toronto technical report CRGTR962* 6, pp. 1–6.

Ghahramani Z. and G. E. Hinton (2000). "Variational Learning for Switching State-Space Models". In: *Neural Compuation*.

Ghahramani Z. and S. T. Roweis (1999). "Learning Nonlinear Dynamical Systems Using an EM Algorithm". In: *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*.

Gilks W. and C. Berzuini (2001). "Following a moving target – Monte Carlo inference for dynamic Bayesian models". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.

Goodfellow I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative Adversarial Nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems*.

Goyal A., A. Sordoni, M.-A. Côté, N. Ke, and Y. Bengio (2017). "Z-Forcing: Training Stochastic Recurrent Networks". In: *Advances in Neural Information Processing Systems 30*.

Graves A. (2013). "Generating Sequences With Recurrent Neural Networks." In: *arXiv preprint arXiv:1308.085*.

Graves A., G. Wayne, and I. Danihelka (2014). "Neural Turing Machines". In: *arXiv:1410.5401*.

Graves A., G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis (2016). "Hybrid computing using a neural network with dynamic external memory". In: *Nature*.

Gregor K., I. Danihelka, A. Graves, and D. Wierstra (2015). "DRAW: A recurrent neural network for image generation". In: *ICML*.

Gu S., Z. Ghahramani, and R. E. Turner (2015). "Neural Adaptive Sequential Monte Carlo". In: *NIPS*, pp. 2611–2619.

Gulrajani I., K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville (2016). "PixelVAE: A Latent Variable Model for Natural Images". In: *arXiv preprint arXiv:1611.05013*.

Gupta S., J. Davidson, S. Levine, R. Sukthankar, and J. Malik (2017a). "Cognitive mapping and planning for visual navigation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Gupta S., D. Fouhey, S. Levine, and J. Malik (2017b). "Unifying Map and Landmark based Representations for Visual Navigation". In: *arXiv:1712.08125*.

Haarnoja T., A. Ajay, S. Levine, and P. Abbeel (2016). "Backprop KF: Learning Discriminative Deterministic State Estimators". In: *NIPS*.

Higgins I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner (2017a). "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In:

Higgins I., A. Pal, A. A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner (2017b). "DARLA: Improving Zero-Shot Transfer in Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*.

Hochreiter S. and J. Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667.

Jang E., S. Gu, and B. Poole (2016). "Categorical Reparameterization with Gumbel-Softmax". In: *arXiv preprint arXiv:1611.01144*.

Johnson M. J., D. Duvenaud, A. B. Wiltschko, S. R. Datta, and R. P. Adams (2016). "Composing graphical models with neural networks for structured representations and fast inference". In: *NIPS*.

Jordan M. I., Z. Ghahramani, T. S. Jaakkola, and L. K. Saul (1999). "An introduction to variational methods for graphical models". In: *Machine Learning* 37.2, pp. 183–233.

Julier S. J. and J. K. Uhlmann (1997). "A New Extension of the Kalman Filter to Nonlinear Systems". In:

Kalman R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME – Journal of Basic Engineering*.

Kantas N., A. Doucet, S. S. Singh, M. Jan, and N. Chopin (2015). "On Particle Methods for Parameter Estimation in State-Space Models". In: *Statistical Science*.

Karl M., M. Soelch, J. Bayer, and P. van der Smagt (2017). "Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data". In: *ICLR*.

King S. and V. Karaiskos (2013). "The Blizzard Challenge 2013". In: *The Ninth Annual Blizzard Challenge*.

Kingma D. and M. Welling (2014). "Auto-encoding variational Bayes". In: *ICLR*.

Kingma D. P., D. J. Rezende, S. Mohamed, and M. Welling (2014). "Semi-Supervised Learning with Deep Generative Models". In: *Proceedings of the International Conference on Machine Learning*.

Kingma D. P., T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling (2016). "Improved Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems*.

Kingma D. and J. Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv:1412.6980*.

Krishnan R. G., U. Shalit, and D. Sontag (2015). "Deep Kalman Filters". In: *arXiv:1511.05121*.

Krishnan R., U. Shalit, and D. Sontag (2017). "Structured Inference Networks for Nonlinear State Space Models". In: *AAAI*.

L. G. Ungerleider and L. G. Haxby (1994). ""What" and "where" in the human brain". In: *Curr. Opin. Neurobiol.* 4, pp. 157–165.

Labbe R. (2015). *Kalman and Bayesian Filters in Python*. https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python.

Larochelle H. and I. Murray (2011). "The Neural Autoregressive Distribution Estimator". In: *AISTATS 2011*.

Le T. A., M. Igl, T. Jin, T. Rainforth, and F. Wood (2018). "Auto-Encoding Sequential Monte Carlo". In: *ICLR*.

LeCun Y. and C. Cortes (2010). "MNIST handwritten digit database". In:

Lenz I., R. A. Knepper, and A. Saxena (2015). "DeepMPC: Learning Deep Latent Features for Model Predictive Control." In: *Robotics: Science and Systems*.

Leonard J. J. and H. F. Durrant-Whyte (1991). "Mobile robot localization by tracking geometric beacons". In: *IEEE Trans. Robotics and Automation*.

Levine S. and P. Abbeel (2014). "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics". In: *Advances in Neural Information Processing Systems 27*.

Li C., J. Zhu, and B. Zhang (2016). "Learning to Generate with Memory". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML*.

Li Y. and S. Mandt (2018). "A Deep Generative Model for Disentangled Representations of Sequential Data". In: *arXiv preprint arXiv:1803.02991*.

Li Y. and R. E. Turner (2016). "RéNyi Divergence Variational Inference". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16.

Lin W., M. E. Khan, and N. Hubacher (2018). "Variational Message Passing with Structured Inference Networks". In: *ICLR*.

Linderman S., M. Johnson, A. Miller, R. Adams, D. Blei, and L. Paninski (2017). "Bayesian Learning and Inference in Recurrent Switching Linear Dynamical Systems". In: *AISTATS*.

Liu D. C. and J. Nocedal (1989). "On the Limited Memory BFGS Method for Large Scale Optimization". In: *Math. Program.*

Liu H., H. Bai, L. He, and Z. Xu (2017). "Stochastic Sequential Neural Networks with Structured Inference". In: *arXiv preprint arXiv:1705.08695*.

Liu J. and R. Chen (1998). "Sequential Monte Carlo methods for dynamic systems". In: *Journal of the American Statistical Association* 93.443, pp. 1032–1044.

Liu Z., P. Luo, X. Wang, and X. Tang (2015). "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*.

Maaløe L., M. Fraccaro, and O. Winther (2017). "CaGeM: A Cluster Aware Deep Generative Model". In: *NIPS Workshop on Advances in Approximate Bayesian Inferences*.

Maaløe L., C. K. Sønderby, S. K. Sønderby, and O. Winther (2016). "Auxiliary Deep Generative Models". In: *Proceedings of the International Conference on Machine Learning*.

Maddison C. J., J. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. Teh (2017a). "Filtering Variational Objectives". In: *Advances in Neural Information Processing Systems 30*.

Maddison C. J., A. Mnih, and Y. W. Teh (2017b). "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables". In: *ICLR*.

Mcgee L. and S. F. Schmidt (1985). *Discovery of the Kalman filter as a practical tool for aerospace and industry*.

McHutchon A. (2014). "Nonlinear Modelling and Control using Gaussian Processes". PhD thesis.

Miller A. H., A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston (2016). "Key-Value Memory Networks for Directly Reading Documents". In: *EMNLP*.

Miller A. C., N. J. Foti, and R. P. Adams (2017). "Variational Boosting: Iteratively Refining Posterior Approximations". In: *Proceedings of the 34th International Conference on Machine Learning*, pp. 2420–2429.

Mnih A. and K. Gregor (2014). "Neural variational inference and learning in belief networks". In: *arXiv:1402.0030*.

Mnih V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *ICML*.

Murphy K. P. (1998). *Switching Kalman Filters*. Tech. rep.

Murphy K. P. (2012). *Machine Learning: A probabilistic perspective*. ISBN: 9780262018029.

Naesseth C. A., S. W. Linderman, R. Ranganath, and D. M. Blei (2018). "Variational Sequential Monte Carlo". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Oh J., V. Chockalingam, S. P. Singh, and H. Lee (2016). "Control of Memory, Active Perception, and Action in Minecraft". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML*.

Oh J., X. Guo, H. Lee, R. L. Lewis, and S. Singh (2015). "Action-Conditional Video Prediction using Deep Networks in Atari Games". In: *NIPS*.

Opper M., M. Fraccaro, U. Paquet, A. Susemihl, and O. Winther (2015). "Perturbation Theory for Variational Inference". In: *NIPS Workshop on Advances in Approximate Bayesian Inferences*.

Paisley J. W., D. M. Blei, and M. I. Jordan (2012). "Variational Bayesian Inference with Stochastic Search". In: *ICML*.

Papamakarios G., I. Murray, and T. Pavlakou (2017). "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems 30*.

Paquet U. and M. Fraccaro (2016). "An efficient implementation of Riemannian Manifold Hamiltonian Monte Carlo for Gaussian Process Models". In: *Technical report*.

Parisotto E. and R. Salakhutdinov (2018). "Neural Map: Structured Memory for Deep Reinforcement Learning". In: *ICLR*.

Patraucean V., A. Handa, and R. Cipolla (2015). "Spatio-temporal video autoencoder with differentiable memory". In: *arXiv:1511.06309*.

Pritzel A., B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell (2017). "Neural Episodic Control". In: *Proceedings of the 34th International Conference on Machine Learning*. Proceedings of Machine Learning Research.

Rabiner L. R. (1990). "Readings in Speech Recognition". In: ed. by A. Waibel and K.-F. Lee. Chap. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.

Racanière S., T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra (2017). "Imagination-Augmented Agents for Deep Reinforcement Learning". In: *Advances in Neural Information Processing Systems 30*.

Raiko T. and M. Tornio (2009). "Variational Bayesian Learning of Nonlinear Hidden State-space Models for Model Predictive Control". In: *Neurocomputing*.

Raiko T., H. Valpola, M. Harva, and J. Karhunen (2007). "Building blocks for variational Bayesian learning of latent variable models". In: *Journal of Machine Learning Research* 8, pp. 155–201.

Ranganath R., D. Tran, and D. M. Blei (2015). "Hierarchical Variational Models". In: *arXiv preprint arXiv:1511.02386*.

Rauch H. E., C. T. Striebel, and F. Tung (1965). "Maximum Likelihood Estimates of Linear Dynamic Systems". In: *Journal of the American Institute of Aeronautics and Astronautics* 3.

Rezende D. J., S. Mohamed, and D. Wierstra (2014). "Stochastic backpropagation and approximate inference in deep generative models". In: pp. 1278–1286.

Rezende D. J. and S. Mohamed (2015). "Variational Inference with Normalizing Flows". In: *Proceedings of the International Conference on Machine Learning*.

Roweis S. and Z. Ghahramani (1999). "A unifying review of linear Gaussian models". In: *Neural Computation* 11.2, pp. 305–45.

Rumelhart D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature*.

Salimans T., D. P. Kingma, and M. Welling (2015). "Markov Chain Monte Carlo and Variational Inference: Bridging the Gap". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France.

Santoro A., S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap (2016). "One-shot Learning with Memory-Augmented Neural Networks". In: *arXiv:1605.06065*.

Särkkä S. (2008). "Unscented Rauch–Tung–Striebel Smoother". In: *IEEE Transactions on Automatic Control*.

Särkkä S. (2013). *Bayesian Filtering and Smoothing*. Cambridge University Press.

Seeger M. W., D. Salinas, and V. Flunkert (2016). "Bayesian Intermittent Demand Forecasting for Large Inventories". In: *NIPS*.

Shabanian S., D. Arpit, A. Trischler, and Y. Bengio (2017). "Variational Bi-LSTMs". In: *arXiv preprint arXiv:1711.05717*.

Shi W., J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang (2016). "Real-time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network". In: *CVPR*.

Shumway R. H. and D. S. Stoffer (1982). "An approach to time series smoothing and forecasting using the EM algorithm". In: *Journal of Time Series Analysis*.

Smith G., S. Schmidt, L. McGee, U. S. N. Aeronautics, and S. Administration (1962). *Application of Statistical Filter Theory to the Optimal Estimation of Position and Velocity on Board a Circumlunar Vehicle*. NASA technical report. National Aeronautics and Space Administration.

Smith R., M. Self, and P. Cheeseman (1987). "Estimating uncertain spatial relationships in robotics". In: *Proceedings. 1987 IEEE International Conference on Robotics and Automation*.

Sønderby C. K., T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther (2016a). "How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks". In: *arXiv:1602.02282*.

Sønderby C. K., T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther (2016b). "Ladder Variational Autoencoders". In: *Advances in Neural Information Processing Systems 29*.

Srivastava N., E. Mansimov, and R. Salakhudinov (2015). "Unsupervised Learning of Video Representations using LSTMs". In: *ICML*.

Sukhbaatar S., A. Szlam, J. Weston, and R. Fergus (2015). "End-to-end Memory Networks". In: *Neural Information Processing Systems*. NIPS'15.

Sun W., A. Venkatraman, B. Boots, and J. A. Bagnell (2016). "Learning to Filter with Predictive State Inference Machines." In: *ICML*.

Sutskever I., O. Vinyals, and Q. V. Le (2014). "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27*.

Sutton R. S. (1990). "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *In Proceedings of the Seventh International Conference on Machine Learning*.

Taketomi T., H. Uchiyama, and S. Ikeda (2017). "Visual SLAM algorithms: a survey from 2010 to 2016". In: *IPSJ Transactions on Computer Vision and Applications* 9, pp. 1–11.

Valpola H. and J. Karhunen (2002). "An Unsupervised Ensemble Learning Method for Nonlinear Dynamic State-Space Models". In: *Neural Computation*.

Van Den Oord A., N. Kalchbrenner, and K. Kavukcuoglu (2016). "Pixel Recurrent Neural Networks". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. ICML'16.

van der Merwe R., A. Doucet, N. De Freitas, and E. Wan (2000). "The Unscented Particle Filter". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'00.

Wahlström N., T. B. Schön, and M. P. Deisenroth (2015). "From Pixels to Torques: Policy Learning with Deep Dynamical Models". In: *arXiv:1502.02251*.

Wan E. A. and R. van der Merwe (2001). "The Unscented Kalman Filter". In: *Kalman Filtering and Neural Networks*. Wiley.

Watter M., J. Springenberg, J. Boedecker, and M. Riedmiller (2015). "Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images". In: *NIPS*.

Winn J. and C. M. Bishop (2005). "Variational Message Passing". In: *Journal of Machine Learning Research*.

Wu J., I. Yildirim, J. J. Lim, W. T. Freeman, and J. B. Tenenbaum (2015). "Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning". In: *NIPS*.

Zaremba W. and I. Sutskever (2015). "Reinforcement Learning Neural Turing Machines". In: *arXiv:1505.00521*.

Zhang J., L. Tai, J. Boedecker, W. Burgard, and M. Liu (2017). "Neural SLAM: Learning to Explore with External Memory". In: *arXiv:1706.09520*.

Zheng X., M. Zaheer, A. Ahmed, Y. Wang, E. P. Xing, and A. J. Smola (2017). "State Space LSTM Models with Particle MCMC Inference". In: *arXiv preprint arXiv:1711.11179*.