DTU Compute
Department of Applied Mathematics and Computer Sciences

Recurrent Flows for Video Generation

Christian Dandanell Glissov (s146996)
Tobias Gylling Konradsen (s144077)

Supervisors: Filipe Rodrigues, Daniele Gammelli, Mathias Niemann Tygesen

M.Sc. Thesis
DTU 2021

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

Density estimation in high-dimensional video data is a complex problem. The high uncertainty involved in predicting future frames, where several possible outcomes are probable, requires flexible density estimators. To enable density estimation in high-dimensional video data, we propose applying a *Recurrent Flow Network*. The model applies a Stochastic Recurrent Neural Network in unison with a normalising flow for efficient sampling and an improved conditional likelihood estimate. The Recurrent Flow Network takes advantage of deterministic and stochastic recurrent hidden states to model highly structured spatio-temporal stochastic data. It has been shown to deliver promising results on low-dimensional urban topologies, but a generalisation to high-dimensional video data is missing. This thesis presents the theory behind Recurrent Flow Networks, and the modification of it with a conditional Glow model. It is done by completely re-implementing the architecture using the deep learning framework PyTorch, making it compatible to a high-dimensional domain, specifically video data. We experiment with different regularisation methods on three data sets and empirically evaluate the model against other recent generative models. Results show that the model is able to estimate complex high-dimensional distributions. Using only a short series of context frames, it is possible to generate realistic predictions that is comparable to competing methods. Lastly, we conduct an ablation study with focus on the stochastic dynamics and density estimation of the recurrent flow network. The study is able to highlight significant pitfalls of the recurrent flow network. It also discovers an interesting property of disentanglement between the temporal and spatial stochastic dynamics.

Preface

This master thesis on 30 ECTS was prepared at the Department of Technology, Management and Economics at the Technical University of Denmark. It was prepared during the autumn semester of 2020 and finished in February 2021 in fulfilment of the requirements for acquiring a master degree in Mathematical Modelling and Computation.

The thesis was supervised by Filipe Rodrigues, associate professor, Daniele Gammelli, PhD student, and Mathias Niemann Tygesen, PhD student, from the Department of Technology, Management and Economics (DTU Management).

DTU, May 26, 2021

Christian D. Glissov (s146996)

Tobias G. Konradsen (s144077)

Acknowledgements

We would like to thank Filipe Rodrigues, Daniele Gammelli and Mathias Niemann Tygesen for excellent guidance during the entire preparation of this thesis. We both agree that this thesis has been a tough, but fruitful, journey into the world of generative modelling.

To the main author of VideoFlow, Manoj Kumar. Thank you for assisting us with the evaluation methods. It was much appreciated.

We would also like to thank Anna Lia Scharling Tromer Dragsdahl for proofreading and supplying us with rations.

Tobias Konradsen would like to make a special acknowledgement to his parents, Per Konradsen and Janne Konradsen. Thank you for supporting me. Also a special thanks to Christian for being an excellent partner during the making of this thesis and an even better friend.

Christian would like to give a special thanks to his family and friends. My heartfelt thanks to you all for the wonderful support and immense patience with me during an otherwise lonely pandemic. Also, of course, a special thank you to my good friend and partner in crime, Tobias.

Contents

Abbreviations

ActNorm	Activation Normalization
AVI	Amortized Variational Inference
BPP	Bits-per-pixel
CNF	Conditional Normalizing Flow
DLVM	Deep Latent Variable Model
DMOL	Discretized mixture of logistics
DSSM	Deep State Space Model
ELBO	Evidence Lower Bound
FID	Fréchet Inception Distance
FVD	Fréchet Video Distance
LPIPS	Learned Perceptual Image Patch Similarity
MSE	Mean-Square Error
PGM	Probabilistic Graphical Model
PSNR	Peak signal-to-noise ratio
VRNN	Variational Recurrent Neural Network
VI	Variational Inference
RFN	Recurrent Flow Network
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SSIM	Structural Similarity Index Measure
SRNN	Stochastic Recurrent Neural Network
SVG	Stochastic Video Generation

CHAPTER 1

Introduction

In recent years, the field of machine learning has seen a large increase in popularity. An exponential progress of computational power and extensive research has contributed to the rise of machine learning applications within several areas. Machine learning is changing the world by transforming most segments including healthcare services, education, transport, entertainment, and the internet. Nowadays most people use machine learning every day without realising; on their phones, when they shop or even when they watch TV. Commonly, the practical applications of machine learning technology have been by supervised learning methods. Supervised methods learn a mapping from an input to an output using labelled data, but the methods are often constrained due to data scarcity. Most of the data collected nowadays are unlabelled, obtained from smart devices and the internet. Therefore, an appealing alternative to supervised learning would be to take advantage of unsupervised learning by using the large quantities of unlabelled data.

A large effort for progress in generative modelling methods has made it possible to estimate complex underlying structures of unlabelled data, e.g. patterns, clusters, correlation, and causal structures. It has led to significant improvements in areas such as classification of images, machine translation, and super-human game-playing agents, among others [1]. By building an internal representation of the data it is possible to generate new data, such as images or video [2]. One challenging task of generating probabilistic models is modelling the variability of the data. In the context of video, variability is what gives a video its quirks, uniqueness and brings diversity into the frame. This variability is often approximated by inference models such as variational autoencoders. An approximated variability can lead models to improperly represent the data they were trained on. To give a better approximation of the underlying variability in the data, it has been shown that separating the deterministic and stochastic states can be beneficial [3].

There is high uncertainty within video data, as a short series of observations, can lead to an infinite series of probable outcomes. Therefore, a model needs to build up an internal representation of the world to model this form of variability. A model learning about a wide range of real-world phenomena can be useful in many areas, such as robotics, modelling physical interactions, and downstream tasks [4]. In this thesis we study the problem of stochastic prediction and will specifically focus on con-

ditional video prediction. The aim is to be able to generate raw RGB video frames conditioned on only a context of few frames. To improve conditional video prediction, flexible models are required, and must take the stochastic and deterministic nature of the world into account.

Normalising flows are a type of flexible models. Here an approximation of the variability can be avoided, as normalising flows give an exact measure of the variability in the data. By modelling the exact underlying distribution, normalising flows make it possible to have models that better represent the data it was trained on and as such the underlying distribution describing a realistic generation problem.

This thesis seeks to explore and elaborate on a generative model for generating temporal data by using a Recurrent Flow Network (RFN)[5]. The RFN separates and approximates the stochastic and deterministic temporal distribution of data with a stochastic recurrent neural network (SRNN)[3], while exactly modelling the spatial variability of data with a normalising flow. The RFN has previously only been tested on sequential 2D data. In this work, the RFN is generalised to higher dimensions, i.e. video data. To do this we believe a convolution based SRNN model [3] together with a conditional Glow model can be successfully applied [6, 7]. The combination is believed to make the RFN able to extrapolate video frames conditioned on a short context of past frames and generate diverse stochastic futures. Thus, this thesis seeks to test the above premise by:

- Introducing the rich theory behind Recurrent Flow Networks.
- Generalising the RFN to video data by re-implementing and extending the architecture of the RFN using the deep learning framework PyTorch.
- Evaluating generative video models on the task of spatio-temporal modelling using different video data.
- Comparing the RFN with state-of-the-art models.
- Conducting an ablation study focusing on regularisation methods and properties of the RFN.

All code is stored on `GitHub` and publicly available here: link.

Overview

To give the reader an understanding of the theory behind RFNs, a general introduction of each sub-field is presented. In Chapter 2 the reader is first introduced to a conceptual understanding of generative models, giving an informal introduction to probabilistic latent based models. This understanding is expanded to deep latent variable models in Chapter 3, which incorporates temporal data modelling and the introduction of the SRNN model. A general introduction to the world of normalising flows and how to condition them on external information is presented in Chapter 4. This chapter is a primer to understand the theory behind a conditional Glow. Together the theoretic background should be set and the information from all of the chapters enable us to dive into the general theory of the RFN in Chapter 5. As the main task of the thesis is to generate quality video, Chapter 6 will explain how to properly evaluate generated video sequences and introduce common methods used to do so. In Chapter 7 the video data is presented. In Chapter 8 the architecture of the implemented models is explained; how to construct a convolution based RFN model by unifying a convolution based SRNN and a conditional Glow. The implemented model is then experimented with and analysed in Chapter 9 and 10. With Chapter 11 elaborating on the results and discussing the advantages and drawbacks of the RFN. Finally, in Chapter 12 a conclusion of the work is drawn.

CHAPTER 2

A Conceptual Understanding

Before delving into the theoretical aspects of unsupervised learning the reader is presented to some basic concepts of unsupervised learning. Section 2.1 will introduce the concept of unsupervised learning. Section 2.2 introduces the concept of learning by generation and how to interpret data using inference. Finally, the chapter will end by talking about the difference between deterministic and stochastic paradigms in section 2.3.

2.1 Unsupervised Learning

Unsupervised learning is essential for a system to be considered intelligent. It enables humans to learn the world by observing it, e.g. that objects move independently of each other, phrases are not mere sound frequencies and how to predict an outcome in the near future. The predictive ability of humans is what gives a ”common sense” and enables us to discover regularities in the world. In artificial systems, such as modern computer systems, the lack of unsupervised learning is what renders a computer as naive. A computer is still considered unable to recognise its environment even when a camera is connected to it. Without learning or human interaction the computer lacks the ability of doing anything by itself [8].

The idea of unsupervised learning is to detect predictable structure in unlabelled data. The fundamental principle of unsupervised learning is that by observing a stream of data, such as natural images, it is possible to derive a model of the relations between data variables, for images these variables would be pixels. In most natural images neighbouring pixels are correlated, this information can be used by unsupervised models to discover patterns and deduct complex inter-variable relations and higher level patterns, such as shapes, colours and positions [8].

In unsupervised learning the data is only observed[1]. Just by observing data it is possible to interpret and to learn a generic abstraction, e.g. recognising the difference between zebras and horses in images. Unsupervised learning contrasts to supervised learning where labelled data is used. In supervised learning the algorithm is taught the difference between zebra and horse by passing correct labels of each image. This leads to a very specific function, optimised for a single task, unable to adapt to changes and limited by the specified labels. Unsupervised learning is commonly used to overcome these problems, but unlike supervised learning the output is not a simple value. The output is instead an internal distributed representation consisting of several variables, which can be difficult to interpret.

In this thesis, unsupervised learning will be used for density estimation by defining a probabilistic algorithm to model stochastic data. The data will not be a single image, but several frames of images over time, i.e. sequential data. There will not only be correlations between the pixels, but also auto-correlations between the current and past frames, i.e. spatio-temporal dependencies. This will significantly increase the complexity of the problem.

2.2 Generation and Inference

For learning to take place the data first needs to be broken down into features. Features will make the problem at hand easier by extracting relevant information contained within the data set, for example by learning to differentiate between two different numbers in an image is easier, when given variables representing lines and curves instead of individual original pixel colours. The ideal high- and low-level features will be disentangled, meaning that the mutual information between the variables are minimised. This means that there should be little redundancy in the feature representation and the feature representation should retain as much of the information of the data as possible [9]. A way to think about features is to think that the data lies on a manifold with a dimensionality much smaller than the original data space. This is also called *The Manifold Hypothesis* [10]. The Manifold Hypothesis states that real-world high-dimensional data lie on low-dimensional manifolds embedded within the high-dimensional space. It is the basis for manifold learning and it is this low-dimensional manifold that is being modelled by the high-level features.

One way to get a good feature representation is to use the learned features to generate a datum; Good representations should give an adequate reconstruction of the observation. To quote Geoffrey Hinton, *"To recognise shapes, first learn to generate images"* [11].

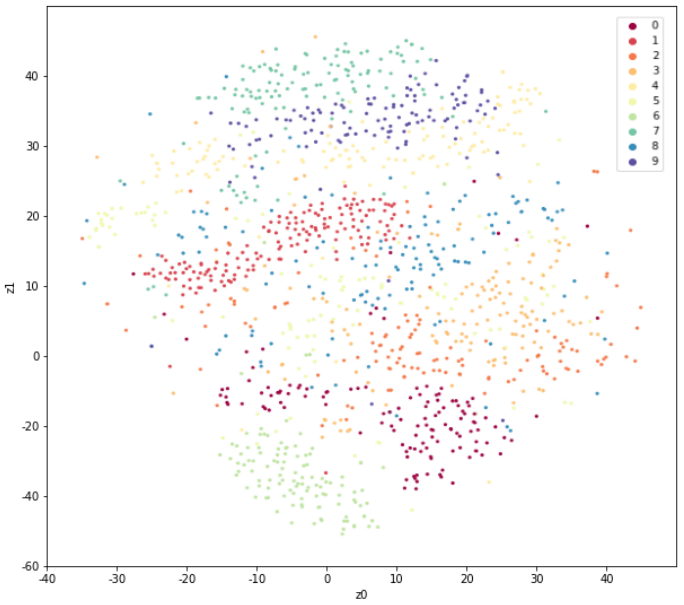
[1] In the context of unsupervised learning, observed data is the unaltered raw data that is fed to the model.

In this thesis, generative latent variable models will be used to derive latent variables. Latent variables are variables that are unobserved, but are inferred from the observed data, hence they represent the underlying variables of the data. The latent variables contain underlying features of the data. A latent variable model will try to learn the latent variables while trying to satisfy the conditions required for a good representation of the data: low redundancy and the ability to reconstruct.

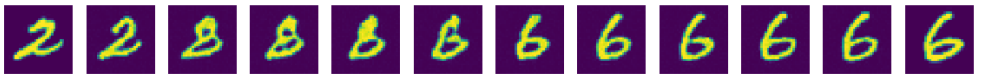
To interpret the datum it is necessary to find the high-level representation that generated it, finding this representation is done by inference. To find the best matching underlying cause for generation of the datum it is not possible to use a naive approach, such as trying all possible latent variable configurations, this is intractable. In most cases, except in very simple models, it is also not possible to invert the generation process, due to multiple non-linearities and layers usually present in deep models [8]. Finally, as in practice high-level representations might not be independent, as such multiple representations could have generated the sample, this renders inference as an ill-defined problem.

A range of problems arise when performing inference. A solution to the problem that renders inference intractable, is to train an inference model, which to some degree resembles the inverted generative model. This bottom-up structure allows the network to determine a plausible explanation to how the model might have generated an observed datum [12]. This means the inference model is trained to produce high-level representations. The generative model will reconstruct these high-level representations back into the data fed to the model, satisfying the reconstructability criterion of generative models. Models applying such dual learning approach are a type of algorithms, namely autoencoders. The theory behind these algorithms will be extensively used throughout this thesis and a more thorough introduction to the theory behind them will be given in Chapter 3.

To illustrate the above stated problems connected to inference, an example is shown in Figure 2.1: The figure (a) shows the latent representations. Notice how the latent representations are grouped in different distributions for each of the digits. Clearly, the latent variables contain relevant information about the underlying data distribution. The latent variables are obtained by passing the images through an inference model. The Glow model will be introduced later in the thesis and for this example it is trained on MNIST data (Chapter 7). As the latent space is high-dimensional, a dimension reduction algorithm (TSNE, [13]) is used to map it to a 2D space. The figure (b) shows an interpolation between two latent variables ($\mathbf{z}_{\text{start}}$, $\mathbf{z}_{\text{end}}$). By doing an interpolation and passing them through the generative part of the model it is possible to see how traversing the latent space generates different digits.



(a) Projection of latent space, using a TSNE algorithm.



(b) Interpolating in latent space.

Figure 2.1: (a) Latent representations of different MNIST digits from 0 to 9. (b) Interpolation between the two digits (2 and 6) using the latent variables ($\mathbf{z}_{\text{start}}$, $\mathbf{z}_{\text{end}}$).

2.3 Deterministic and Stochastic Modelling

In unsupervised learning methods there are often two paradigms that need to be taken into account. The first is deterministic. Here exact values are computed for the estimated variables and the general observed structure of the data is modelled. The second is probabilistic where variables come from a distribution of values and allows for modelling variability and uncertainty within data. A probabilistic method can be regarded as a probabilistic interpretation of some underlying deterministic method that includes a noise source and allows for a framework with a more abstract problem statement.

In generative models, probabilistic and deterministic approaches differ. Taking a deep autoencoder as an example, it is seen that in the deterministic case the generation process contains a decoder parameterised by a deep neural network, hence all the values are exact for each of the layers in the decoder. For the deterministic generative

case, the intermediate and bottom layers consist of fully dependent variables that is uniquely determined by a top layer. Further, in the deterministic case every detail of a generated item must be represented in the high-level representation. As a result, the inference model is not able to store unique details in each of the layers when moving up in the bottom-up structure, ultimately leaving the model with a flat representation. The inflexibility means that when a small change in an image happens, then the change has to correspond to a change in the high-level representation of the image. An intractable amount of hidden units might be required, when taking all the possible changes into account [14].

To prevent the issue with a deterministic generation it is required that the representation contains abstract features. A probabilistic generative model will allow exactly this. In probabilistic generative models a distribution assumption is used that enables a layer to produce any value from a distribution of values. Often the model first estimates an expected value and then adds noise. The probabilistic generative model allows for variation in deep neural networks. It allows for high-level representations to manifest themselves in different ways, each with a unique probability of occurring. In Bayesian statistics, a probabilistic generative model also defines a prior probability distribution. For inference the prior is important, as it is used to tell from which representation the datum has been generated. This is done by assigning a prior probability to each probable cause. Every layer having the freedom to pick from a distribution of values making the top layer contain an abstract high-level representation. This freedom allows lower layers to be responsible for different aspects within an image assigned by the inference. Combined, this abstract representation of the top layer and the more concrete details of the lower levels, form a deep representation [8]. The implication of using deterministic layers to model data with uncertainty can be seen in Figure 2.2.

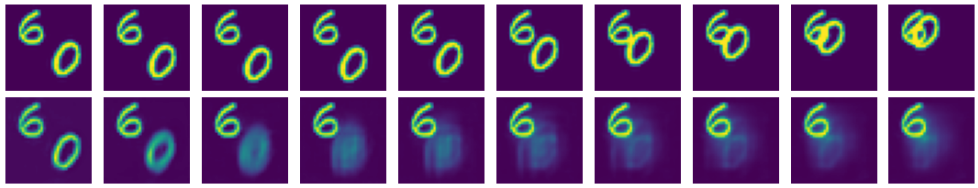


Figure 2.2: First row illustrates the target sequence. Second row illustrates predictions for stochastic moving MNIST data passed through a deterministic autoencoder containing convolutional long-short term memory (ConvLSTM, [15]) cells.

An example is shown in Figure 2.2, where the consequence of processing stochastic data with a deterministic model is shown. The first row of Figure 2.2 illustrates the target sequence, while the second row illustrates the predictions of the model. The stochasticity of the data arises when a digit collides with the border of the image. When colliding with the border the velocity is sampled, being different for each time

a collision happens. In the figure the digit 6, in contrast to the digit 0, has almost no velocity. When 0 is near a boundary the digit will turn blurry. The deterministic model is oblivious to where the digit will move as the bouncing is a stochastic process. The digit 6 is still very clear, as it is constrained in a deterministic setting due to its velocity not changing. In this thesis, models that take into account both the deterministic structure and the stochastic variability of data are used.

CHAPTER 3

Latent Variable Models

This chapter will dive into the more theoretic aspects of probabilistic modelling. The chapter is based on work done by Kingma et al. [16] and Fraccaro et al. [2], it will introduce some key concepts that are used later in this thesis. Section 3.1 will shortly introduce deep latent variable models (DLVM). Section 3.2 will introduce variational inference and a fundamental deep latent variable model; Variational Autoencoder (VAE). Section 3.3 will shortly look at modelling deterministic sequential data. Finally, Section 3.4 will expand on the theory of variational autoencoders by looking at deep latent variable models for sequential data.

3.1 Basics

The central problem of unsupervised learning is to model a complicated marginal probability distribution $p(\mathbf{x})$ that can approximate the true unknown probability distribution $p^*(\mathbf{x})$. Where $\mathbf{x}$ is an observed random sample of high dimensional data $\mathbf{x} \sim p^*(\mathbf{x})$. As shortly described in Section 2.2 the distribution $p(\mathbf{x})$ is not modelled directly, but rather by using latent variables, $\mathbf{z}$, by a conditional distribution $p(\mathbf{x}|\mathbf{z})$. In this case $\mathbf{z}$ is assumed continuous, but mainly the same principles also apply to the discrete case. The conditional distribution $p(\mathbf{x}|\mathbf{z})$ is also called the *conditional likelihood*. The dependence of $\mathbf{x}$ given $\mathbf{z}$ makes it possible to express the correlations in the observed data defining high-level features of images i.e. colour, size, position, curves, by modelling $\mathbf{z}$. The joint distribution $p(\mathbf{x}, \mathbf{z})$ over observed and latent variables are used to define a more tractable expression for the marginal distribution $p(\mathbf{x})$ by definition

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}). \tag{3.1}$$

Here it is possible to define the *prior distribution* $p(\mathbf{z})$ and the likelihood $p(\mathbf{x}|\mathbf{z})$ as simple distributions[1]. To then define the generative part of the model it is necessary to

[1] Often a distribution from the exponential family is assumed because of the prior being conjugate to the posterior, their simple analytical expressions, and the functional form. This simplifies the computation and interpretation when doing Bayesian inference.

marginalise out the latent variable. For the continuous case this is done by integrating over $\mathbf{z}$:

$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z} = \int p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) \mathrm{d}\mathbf{z}. \tag{3.2}$$

Finally, inference can be obtained by using Bayes' rule as

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{p(\mathbf{x})}, \tag{3.3}$$

where $p(\mathbf{z}|\mathbf{x})$ is commonly known as the *posterior distribution*. It is often attempted to approximate the underlying process $p(\mathbf{x})$ by using parametric families of distributions. The notation of the joint distribution in Equation 3.1 can be explicitly written as

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

where $\boldsymbol{\theta}$ are *learned* unknown parameters of the model based on the given data. Often the approximate distribution is based on deep neural networks, which also gives rise to deep latent variable models, such as Variational Autoencoders (VAE).

A problem in DVLMs is intractability. The marginal likelihood, $p(\mathbf{x})$, and the posterior, $p(\mathbf{z}|\mathbf{x})$, are both intractable as Equation 3.2 has no analytic solution. One could argue that $p(\mathbf{x})$ can conceptually be approximated by using Monte Carlo integration $p(\mathbf{x}) \approx \frac{1}{n} \sum_i p(\mathbf{x} \mid z_i)$, but n might need to be very large, causing the problem to be infeasible [17]. Due to the intractable distributions it is impossible to differentiate and optimise the parameters of the model. Approximate inference techniques such as variational inference allow for an approximation of the posterior and the marginal likelihood. This leads to an approximate solution to the problem in Equation 3.1.

3.2 Variational Autoencoder

Variational autoencoders are a computationally efficient way to optimise a generative model jointly with an inference model using stochastic gradient descent (SGD) [16]. VAEs use variational inference to approximate the posterior distribution. The variational inference is computationally efficient, it is even able to give exact results if the underlying distribution matches the chosen variational family of distribution. Although, for most data that have not been simulated this is not the case. Real data is complex and as such the true posterior rarely falls inside the classic parametric families. In those cases, variational inference will approximate the posterior.

3.2.1 Variational Inference

To turn the intractable problem of posterior inference into a tractable problem one can use variational inference (VI). The idea of inference is to find latent variables $\mathbf{z}$ that are likely to have produced $\mathbf{x}$ and can be used to compute the marginal distribution

$p(\mathbf{x})$. To do this a new distribution is required $q(\mathbf{z}|\mathbf{x})$ that is able to give a distribution over the latent variables $\mathbf{z}$, but first an arbitrary distribution $q(\mathbf{z})$ is used to see how $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}P(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{x})$ relates to each other. Such is done by firstly defining a way to approximate the posterior distribution $p(\mathbf{z}|\mathbf{x})$ by using a measure of dissimilarity. An often used dissimilarity measure is the *Kullback-Leibler* (KL) divergence. It measures the divergence between the variational distribution $q(\mathbf{z})$ and the posterior distribution [18] by

$$\mathrm{D}_{\mathrm{KL}}\left[q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})\right] = -\int q(\mathbf{z}) \log\left(\frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})}\right) \mathrm{d}\mathbf{z} \tag{3.5}$$

$$= -\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log \frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})}\right]. \tag{3.6}$$

The KL divergence[2] can be between 0 and ∞, with 0 being if and only if $q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x})$, and the D_{KL} being larger the further away $q(\mathbf{z})$ is from $p(\mathbf{z}|\mathbf{x})$.

The intractable posterior distribution $p(\mathbf{z}|\mathbf{x})$ still appears in the numerator of Equation 3.6. To get rid of it, Bayes' rule from Equation 3.3 and the joint distribution $p(\mathbf{x}, \mathbf{z})$ are used to attain

$$\mathrm{D}_{\mathrm{KL}}\left[q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})\right] = -\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} - \log p(\mathbf{x})\right], \tag{3.7}$$

where the marginal log-likelihood can be moved out of the expectation as it does not depend on $\mathbf{z}$. This leaves the expression as

$$\mathrm{D}_{\mathrm{KL}}\left[q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})\right] = \underbrace{-\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})}\right]}_{=\mathcal{F}(q)} + \log p(\mathbf{x}). \tag{3.8}$$

The first term $\mathcal{F}(q)$ in Equation 3.8 is the Evidence Lower Bound (ELBO) and is maximised with respect to q for all $q(\mathbf{z})$. Because of the non-negativity of D_{KL} the ELBO will be a lower bound to the marginal log-likelihood, $\mathcal{L}(\boldsymbol{x})$ this is seen as

$$\mathcal{F}(q) = \log p(\mathbf{x}) - \mathrm{D}_{\mathrm{KL}}\left[q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})\right] \tag{3.9}$$

$$\leq \log p(\mathbf{x}) = \mathcal{L}(\boldsymbol{x}). \tag{3.10}$$

By maximising the ELBO with respect to the distribution $q(\mathbf{z})$ the KL will be minimised in Equation 3.10, which implies a better approximation of the posterior distribution, hence the ELBO will get closer to the marginal likelihood. To make Equation 3.8 resemble the encoder and decoder structure of VAEs it is possible to rearrange and expand the equation using the joint distribution $p(\mathbf{x}, \mathbf{z})$ by

$$\mathrm{D}_{\mathrm{KL}}\left[q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})\right] = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[-\log p(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z}) + \log q(\mathbf{z})\right] + \log p(\mathbf{x}). \tag{3.11}$$

[2]The KL divergence is not symmetric $\mathrm{D}_{\mathrm{KL}}[q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})] \neq \mathrm{D}_{\mathrm{KL}}[p(\mathbf{z})\|q(\mathbf{z}|\mathbf{x})]$, i.e. it is not a distance metric. In VI a mode-seeking KL is the standard.

It is now possible to contract $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}$ into a KL divergence, once again the equation is rearranged

$$\log p(\mathbf{x}) - \mathrm{D}_{\mathrm{KL}}[q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})] = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[\log p(\mathbf{x}|\mathbf{z})] - \mathrm{D}_{\mathrm{KL}}[q(\mathbf{z})\|p(\mathbf{z})]. \tag{3.12}$$

Finally, q will now depend on $\mathbf{x}$ as we are interested in when inferring $p(\mathbf{x})$

$$\log p(\mathbf{x}) - \mathrm{D}_{\mathrm{KL}}[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})] = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - \mathrm{D}_{\mathrm{KL}}[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})]. \tag{3.13}$$

Here the left side contains the log of the marginal distribution that should be maximised, $\log p(\mathbf{x})$ and an error term that is minimised when $q(\mathbf{z}|\mathbf{x})$ produces $\mathbf{z}$ that are good at reconstructing $\mathbf{x}$. The right hand side also resembles a decoder and inference network, q encodes $\mathbf{z}$ from $\mathbf{x}$ and p decodes $\mathbf{z}$ to reconstruct $\mathbf{x}$. In practice, the distributions $q(\mathbf{z}|\mathbf{x})$, $p(\mathbf{z})$, and $p(\mathbf{x}|\mathbf{z})$ are often set to parametric families that are computationally easy to work with, such as the Gaussian distribution.

3.2.2 The generative model

In VAEs the distribution parameters are estimated by deep neural networks. The generative model has the parameters θ and is specified as:

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z}). \tag{3.14}$$

The prior is often given as an isotropic multivariate Gaussian with zero mean and an identity covariance matrix, $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. A Gaussian distribution is often chosen because of its convenient properties of having an analytic KL divergence solution, efficient sampling, and a well-defined reparameterisation trick allowing for efficient gradient computation. The conditional likelihood is the decoder, which is defined by a parametric distribution that matches the data type, e.g. a Bernoulli distribution for binary data or a Gaussian for continuous data. However, the parametric distribution can also be modelled with more sophisticated methods that will be introduced later in this thesis.

Looking at the Gaussian case, $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))$, the parameters of the Gaussian distribution will then be estimated by a deep neural network $(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \mathrm{Decoder}_\theta(\mathbf{z})$, while θ are the weights and bias parameters of the neural nets. By getting a random latent variable sample $\mathbf{z}^{sample}$ from the prior, it is then possible to get a new observation $\mathbf{x}^{sample}$ that can be used to evaluate how well the model approximates the data distribution.

3.2.3 The inference model

As previously mentioned, because of intractability it is necessary to have an inference model to approximate the variational posterior to the true posterior

$$q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x}). \tag{3.15}$$

An issue with variational inference is that for each data point $\{\mathrm{x}^i\}_{i=1}^N$ it is necessary to learn a different set of parameters ϕ_i, this means that the parameters scale linearly with the number of data points. To avoid this, *amortised variational inference* (AVI) [19] is used by introducing a constraint to the number of parameters ϕ. This will introduce a trade-off with higher efficiency but lower expressiveness, as in AVI the parameters ϕ is now shared across all data points. This trade-off is worth it, as there is no longer a linear scaling of parameters and it is possible to leverage the efficiency of stochastic gradient decent (SGD). An encoder consisting of deep neural networks is used to estimate the variational parameters of the distribution, for the Gaussian case this is

$$\begin{aligned}(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) &= \text{ Encoder}_\phi(\mathbf{x}) \\ q_\phi(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})).\end{aligned} \tag{3.16}$$

Again, the parameters ϕ are then the weights and bias of the encoder network. The encoder is chosen so that it will compress the input $\mathbf{x}$ leading to a bottleneck between the encoder and decoder. This ensures only the main structured part of the information will go through, hence the dimension of the latent variables $\mathbf{z}$ will be smaller than the dimension of $\mathbf{x}$. The decoder in the generative model will then use the latent variables to reconstruct the input.

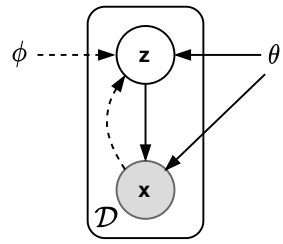


Figure 3.1: A graphical representation of the VAE. Circle nodes are stochastic, with a shaded node being observed and a white node being latent. Dashed arrows indicate the bottom-up inference q_ϕ and solid arrows the top-down generation p_θ. The parameters that are being optimised $\{\theta, \phi\}$ are also listed as conditions, but will be omitted in later graphs to avoid clutter.

3.2.4 Training a VAE Model

The loss given by the maximum log-likelihood found in Equation 3.13 is then optimised using SGD with respect to ϕ and θ in the generative and inference model. The loss function can be decomposed into a reconstruction term that is estimated by the generative model and a regularisation term estimated by the inference model

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\big[\log p_\theta(\mathbf{x}) - \mathrm{D}_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})]\big] = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\big[\underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]}_{\text{Reconstruction term}} - \underbrace{\mathrm{D}_{\mathrm{KL}}\left[q_\phi(\mathbf{z}|\mathbf{x})\|\log p_\theta(\mathbf{z})\right]}_{\text{Regularisation term}}\big], \tag{3.17}$$

where $\mathcal{D}$ is the data distribution from which $\mathbf{x}$ is sampled. The reconstruction term makes sure the generative model does a good job at reconstructing the data points, while the regularisation term prevents the posterior approximate from being too far away from the prior. The parameters ϕ and θ are jointly optimised by using back propagation. The inner expectation of the reconstruction term is found by a Monte Carlo approximation in Equation 3.17. The maximisation problem can then be set up as

$$\sum_i^n \mathcal{F}_i(\theta,\phi) = \sum_i^n \Big(\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x}_i)}\left[\log p_\theta(\mathbf{x}_i \mid \mathbf{z})\right] - \mathrm{D}_{\mathrm{KL}}\left[q_\phi(\mathbf{z} \mid \mathbf{x}_i)\|\log p_\theta(\mathbf{z})\right]\Big), \tag{3.18}$$

where the ELBO $\mathcal{F}_i(\theta,\phi)$ stresses that the problem is a maximisation over the parameters θ and ϕ for data point $\mathbf{x}_i$ [2]. The expectation in Equation 3.18 does not have a closed-form solution as q is a stochastic unit within the network parameterised by ϕ [20].

To make the Monte Carlo estimate of the expectation, $\mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right]$, differentiable with respect to ϕ the reparameterization trick is used. The reparameterization trick changes a gradient of an expectation to an expectation of a gradient. In practice, for a Gaussian distribution assumption, this is done by sampling noise from a standard Gaussian distribution, $\boldsymbol{\epsilon} \sim \mathcal{N}(0,1)$. One can then reparameterize the stochastic latent variable, $\mathbf{z} = \boldsymbol{\epsilon}\boldsymbol{\sigma}_x + \boldsymbol{\mu}_x$ where x indicates that $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are estimated by pushing $\mathbf{x}$ through the encoder. This allows the stochasticity to be "contained" within $\boldsymbol{\epsilon}$ and makes $\mathbf{z}$ act as a deterministic node [16]. This means it is now possible to avoid differentiating w.r.t. sampling of $\mathbf{z}$ and allows the backpropagation to flow through the network.

When the backpropagation can flow through the latent variable it is possible to optimise ϕ and θ. Monte Carlo integration, often done with a single sample for each forward pass, can then be computed to approximate the intractable expectation. For a more thorough explanation of backpropagation in VAEs, see Kingma et al. [16]. The KL divergence can be computed analytically for the Gaussian case, for other distributions numerical integration might be needed, such as for the Student-t distribution [21].

3.2.5 Importance Weighted Autoencoders

A common approach to a strictly tighter log-likelihood lower bound is by sampling K latent variables creating an importance weighting of the estimated log-likelihood. From Burda et al. [22] it is shown that a lower bound to observed log-likelihood can be found, they state the following Theorem [22]:

Theorem 1 *For all k, the lower bounds satisfy*

$$\log p(\mathbf{x}) \geq \mathcal{F}_{k+1} \geq \mathcal{F}_k$$

Moreover, if $p(\mathbf{z}, \mathbf{x})/q(\mathbf{z} \mid \mathbf{x})$ is bounded, then $\mathcal{F}_k$ approaches $\log p(\mathbf{x})$ as k goes to infinity.

This can be achieved by introducing an importance weighted log-likelihood

$$\mathcal{F}_K(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{n} \mathbb{E}_{\mathbf{z}_{i1},\ldots,\mathbf{z}_{iK} \sim q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_\phi(\mathbf{z}_{ik}|\mathbf{x}_i)} \right], \tag{3.19}$$

where $K \in \mathbb{N}$. In this thesis importance weighting of the loss will be used only when evaluating, as it can be very computationally demanding. For a more in depth explanation of importance weighting the reader is referred to Burda et al. [22].

3.3 Recurrent Neural Networks

Up until now only methods used for non-sequential data have been introduced. This section will introduce methods for sequential data, e.g. video, music, speech or text. Sequential data can be a lot more complex than non-sequential data by having complex high-dimensional temporal distributions, long-term temporal dependencies, and the data is often large in scale. To solve these issues it is necessary to have an architecture that is flexible and scalable. In Section 3.2 the VAE was introduced, this is a fairly flexible and scalable model that is able to model complex high-dimensional data.

A recurrent neural network (RNN) is able to model long-term dependencies in the data by using parametric memory cells. By modifying the theory of VAEs and using RNNs it is possible to create deep state space models (DSSM), that allow for scalable and flexible models for sequential data. To reduce the scope of the thesis, no explaining of the interesting background of DSSM will be given, but if curious about the theory one is referred to Fraccaro et al. [2].

Recurrent neural networks are widely used to model sequences of data. It does this by taking in a variable-length sequence $\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ and at each timestep t, the RNN recursively processes the input $\mathbf{x}_t$ and updates the hidden state $\mathbf{h}_t$ by

$$\mathbf{h}_t = f_\theta \left(\mathbf{h}_{t-1}, \mathbf{x}_{t-1} \right), \tag{3.20}$$

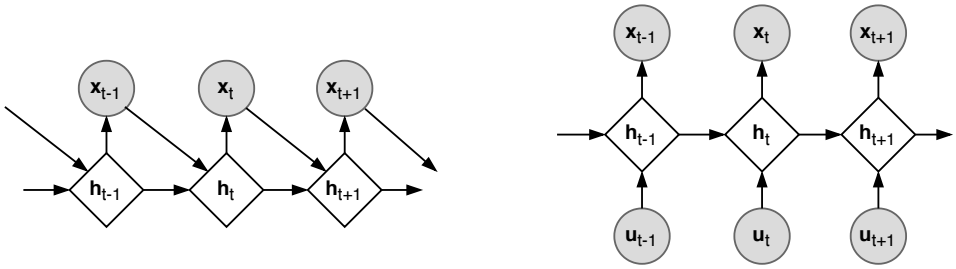


Figure 3.2: A graphical representation of a RNN without (left) and with (right) external covariates. Diamond-shaped nodes are deterministic.

where f is some deterministic differentiable non-linear transition function parameterized by θ and is also called the transition function. The transition function is often implemented using gated activation functions such as long short-term memory (LSTM, [23]) or gated recurrent units (GRU, [24]). The state $\mathbf{h}_t$ will then store information that can help model future time steps. To do sequence modelling, RNNs assume a factorization of the joint distribution over the sequence $\mathbf{x}_{1:T}$,

$$p_\theta\left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\right) = \prod_{t=1}^{T} p\left(\mathbf{x}_t \mid \mathbf{x}_{<t}\right), \tag{3.21}$$

where the conditional distribution is defined by a distribution function with a parameter set θ that depends on the hidden state $\mathbf{h}_t$,

$$p_\theta\left(\mathbf{x}_t \mid \mathbf{x}_{<t}\right) = p_\theta\left(\mathbf{x}_t | \mathbf{h}_{t-1}\right). \tag{3.22}$$

Typically, $\mathbf{h}_0$ is initialised to $\mathbf{0}$. As $\mathbf{h}_t$ is a deterministic unit the RNN will not be able to encode the variability and uncertainty of stochastic data in the deterministic hidden latent states.

It is also possible to model $\mathbf{x}_{1:T}$ that depends on some external covariate $\mathbf{u}_{1:T} = [\mathbf{u}_1, \ldots, \mathbf{u}_T]$. This is simply done by taking into account the added information from $\mathbf{u}_{1:T}$ in Equation 3.20, $\mathbf{h}_t = f_\theta\left(\mathbf{h}_{t-1}, \mathbf{u}_t\right)$. For sequence modelling the condition is often set to $\mathbf{u}_t = \mathbf{x}_{t-1}$. The PGMs of the recurrent neural networks can be seen in Figure 3.2.

3.4 Variational Recurrent Neural Network

In this section the *Variational Recurrent Neural Network* [25] will be introduced. It is a recurrent latent variable model and draws inspiration from dynamic Bayesian network theory. The VRNN is parameterized by deep neural networks, it is a very flexible algorithm and is able to model highly non-linear dynamics and the underlying probabilistic dependencies.

3.4.1 The Generative Model

The VRNN can be interpreted as having a VAE at every time step, which is also seen from Subsection 3.4.3. The difference between a VRNN and a VAE is that the VRNN condition the VAEs on a state variable $\mathbf{h}_t$ of an RNN. This is to take into account the temporal structure of sequential data. In the VRNN the prior is not a standard Gaussian, but follows a Gaussian distribution that is parameterised by a deep neural net

$$\mathbf{z}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{0,t}, \operatorname{diag}\left(\boldsymbol{\sigma}_{0,t}^2\right)\right), \text{ where } \left[\boldsymbol{\mu}_{0,t}, \boldsymbol{\sigma}_{0,t}\right] = \varphi_\theta^{\text{prior}}\left(\mathbf{h}_t\right), \tag{3.23}$$

where $\boldsymbol{\mu}_{0,t}$ and $\boldsymbol{\sigma}_{0,t}$ denote the parameters of the conditional prior distribution. It is then possible to generate $\mathbf{z}_t$ from the Gaussian distribution. The generating distribution will then be conditioned on both $\mathbf{z}_t$ and $\mathbf{h}_{t-1}$

$$\mathbf{x}_t \mid \mathbf{z}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{x,t}, \operatorname{diag}\left(\boldsymbol{\sigma}_{x,t}^2\right)\right), \text{ where } \left[\boldsymbol{\mu}_{x,t}, \boldsymbol{\sigma}_{x,t}\right] = \varphi_\theta^{\text{dec}}\left(\varphi_\theta^{\mathbf{z}}\left(\mathbf{z}_t\right), \mathbf{h}_t\right), \tag{3.24}$$

where $\boldsymbol{\mu}_{x,t}$ and $\boldsymbol{\sigma}_{x,t}$ denote parameters of the generating Gaussian distribution. To help learning complex sequences [25], $\varphi_\theta^{\text{dec}}$ and $\varphi_\theta^{\text{prior}}$ are highly flexible deep neural networks that extract features of their respective input. The same applies to $\varphi_\theta^{\mathbf{u}}$ and $\varphi_\theta^{\mathbf{z}}$, respectively. The idea of not only letting the decoder depend on $\mathbf{z}_t$ but also $\mathbf{h}_t$, is that each of them is better at modelling different aspects of the data. As RNNs are very flexible and powerful architectures, the deterministic state $\mathbf{h}_t$ is great at capturing the overall structure of the data while $\mathbf{z}_t$ are consisting of stochastic latent variables, making it ideal for capturing the variability in the data. Fraccaro et al. [2] gives a great intuitive example of this, illustrating how fundamental such rather subtle change is. When modelling speech the RNNs will model the high-level structure of the wave-forms of the speech, but because the hidden states are deterministic, they will not be able to model nuances and variations across different speakers. Stochastic latent variables are required.

VRNN also assumes that the variability in the data is consistent over time. This is a reasonable assumption, e.g. a speaker will often have the same particular vocal characteristics over time, hence the deterministic state can be defined as

$$\mathbf{h}_t = f_\theta\left(\varphi_\theta^{\mathbf{u}}\left(\mathbf{u}_t\right), \varphi_\theta^{\mathbf{z}}\left(\mathbf{z}_{t-1}\right), \mathbf{h}_{t-1}\right), \tag{3.25}$$

where f is the transition function as explained in Section 3.3. Finally, it is possible to define the joint probability based on the PGM in Figure 3.3, Equation 3.23 and Equation 3.24

$$\begin{aligned} &p_\theta(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0) \\ &\qquad = \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right) p_\theta\left(\mathbf{z}_t \mid \mathbf{h}_t\right) p_\theta\left(\mathbf{h}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_{t-1}, \mathbf{u}_t\right), \end{aligned} \tag{3.26}$$

where $p_\theta\left(\mathbf{h}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_{t-1}, \mathbf{u}_t\right)$ is simply a delta function given by the transition function f as $\mathbf{h}_t$ is deterministic.

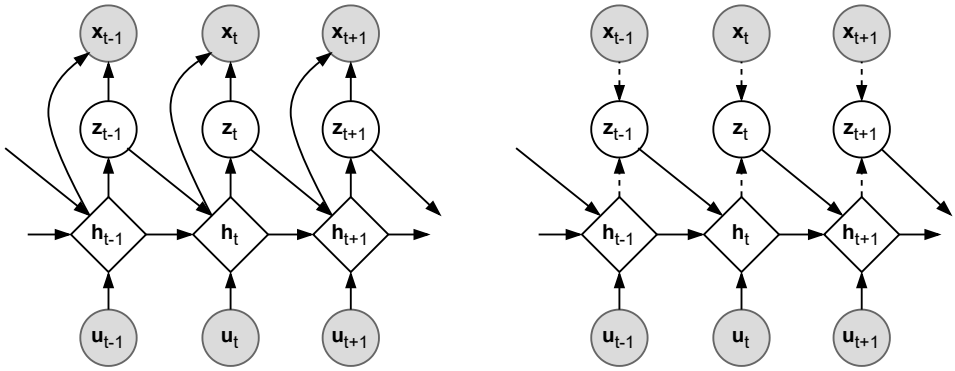


Figure 3.3: A graphical representation of the VRNN. The generative model p_θ can be seen to the left and the inference model q_ϕ to the right.

3.4.2 The Inference Model

The approximate posterior $q\left(\mathbf{z}_{\leq T} \mid \mathbf{x}_{\leq T}\right)$ is a function of $\mathbf{x}_t$ and $\mathbf{h}_t$ and follows the equation

$$\mathbf{z}_t \mid \mathbf{x}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{z,t}, \operatorname{diag}\left(\boldsymbol{\sigma}_{z,t}^2\right)\right), \text{ where } \left[\boldsymbol{\mu}_{z,t}, \boldsymbol{\sigma}_{z,t}\right] = \varphi_\theta^{\text{enc}}\left(\varphi_\theta^{\mathbf{u}}\left(\mathbf{x}_t\right), \mathbf{h}_t\right). \tag{3.27}$$

The conditioning of $\mathbf{h}_t$ will lead to the variational approximation also being conditioned on previous latent variables, leading to the factorization

$$q_\phi\left(\mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{x}_{\leq T}, \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right) = \prod_{t=1}^{T} q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{h}_t\right) p_\theta\left(\mathbf{h}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_{t-1}, \mathbf{u}_t\right), \tag{3.28}$$

where the distribution $p_\theta\left(\mathbf{h}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_{t-1}, \mathbf{u}_t\right)$ is a delta function because $\mathbf{h}_t$ is deterministic. Sampling is efficient and can be done by using ancestral sampling[3] of the normal distribution parameterized by $\boldsymbol{\mu}_{z,t}$ and $\boldsymbol{\sigma}_{z,t}$ that is once again found by using a deep neural network, $\varphi_\theta^{\text{enc}}$.

[3] Ancestral sampling is a process of generating samples from a probabilistic model by first sampling variables without parents using their prior distributions, then sampling their child variables conditioned on the sampled values, then sampling the children's child variables and so on.

3.4.3 Training a VRNN Model

From the joint and variational approximation the objective function can be defined

$$\begin{aligned}\mathcal{F}_i(\theta,\phi) &= \mathbb{E}_{q_\phi(\mathbf{z}_{\leq T},\mathbf{h}_{\leq T}|\mathbf{x}_{\leq T},\mathbf{u}_{\leq T},\mathbf{h}_0,\mathbf{z}_0)}\left[\log \frac{p_\theta\left(\mathbf{x}_{\leq T},\mathbf{z}_{\leq T},\mathbf{h}_{\leq T} \mid \mathbf{u}_{\leq T},\mathbf{h}_0,\mathbf{z}_0\right)}{q_\phi\left(\mathbf{z}_{\leq T},\mathbf{h}_{\leq T} \mid \mathbf{x}_{\leq T},\mathbf{u}_{\leq T},\mathbf{h}_0,\mathbf{z}_0\right)}\right] \\ &= \sum_{t=1}^{T}\Big[\mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{x}_t,\mathbf{h}_t)}(\log p_\theta(\mathbf{x}_t \mid \mathbf{z}_t,\mathbf{h}_t)) - \mathrm{D}_{\mathrm{KL}}(q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t,\mathbf{h}_t\right)\|p_\theta\left(\mathbf{z}_t \mid \mathbf{h}_t\right))\Big].\end{aligned} \tag{3.29}$$

The derivation can be seen in Appendix A.1. Again the same structure for the loss is observed with a reconstruction loss and a KL divergence loss, but this time a summation is done over each time step t. By maximising the ELBO, the KL divergence will be minimised. This enable the possibility of learning the parameters $\{\theta,\phi\}$ by SGD.

3.5 Stochastic recurrent neural networks

Stochastic recurrent neural network (SRNN) is an extension to the VRNN model that was introduced by Fraccaro et al. [3]. The VRNN has a deterministic bottleneck as $\mathbf{h}_t$ depends on the previous stochastic latent variables $\mathbf{z}_{t-1}$. This dependency makes it difficult for the VRNN to properly model how the uncertainty propagates across time steps in the latent variables $\mathbf{z}_t$ [2]. The subtle change of no longer letting $\mathbf{h}_t$ depend on noisy stochastic input from $\mathbf{z}_t$ (i.e. removing the arrow from $\mathbf{z}_{t-1}$ to $\mathbf{h}_t$ in Figure 3.3), but only depend on deterministic input, removes the deterministic bottleneck in the VRNN (See Figure 3.4). Instead $\mathbf{z}_t$ now directly depends on the previous time steps of the latent variables, $\mathbf{z}_{t-1}$. This separates the model into a fully deterministic and stochastic part. It also allows the SRNN to take advantage of principles commonly used in DSSM, such as smoothing, which is explained in the next paragraph. The SRNN will be used as inspiration in Chapter 5 as part of the model *Recurrent Flow Network*.

Fraccaro et al. [3] takes advantage of the temporal structure of the data to improve upon the inference network, by letting the inference network learn from the future of the sequence. It is done by going over the sequence $\mathbf{x}_t$ in reverse, such that the approximate variational posterior gets information of the entire temporal distribution for every time-step. That is to say it not only depends on the past and present, but also the future information. The factorisation is then defined as

$$q_\phi\left(\mathbf{z}_{1:T} \mid \mathbf{h}_{1:T},\mathbf{x}_{1:T},\mathbf{z}_0\right) = \prod_t q_\phi\left(\mathbf{z}_t \mid \mathbf{z}_{t-1},\mathbf{h}_{t:T},\mathbf{x}_{t:T}\right) = \prod_t q_{\phi_\mathbf{z}}\left(\mathbf{z}_t \mid \mathbf{z}_{t-1},\mathbf{a}_t\right), \tag{3.30}$$

where $\mathbf{a}_t = g_\phi\left(\mathbf{a}_{t+1},[\mathbf{h}_t,\mathbf{x}_t]\right)$, and g_ϕ is a RNN propagating backwards through the time series, with $\mathbf{a}_t$ encoding information from the future. Such that the posterior

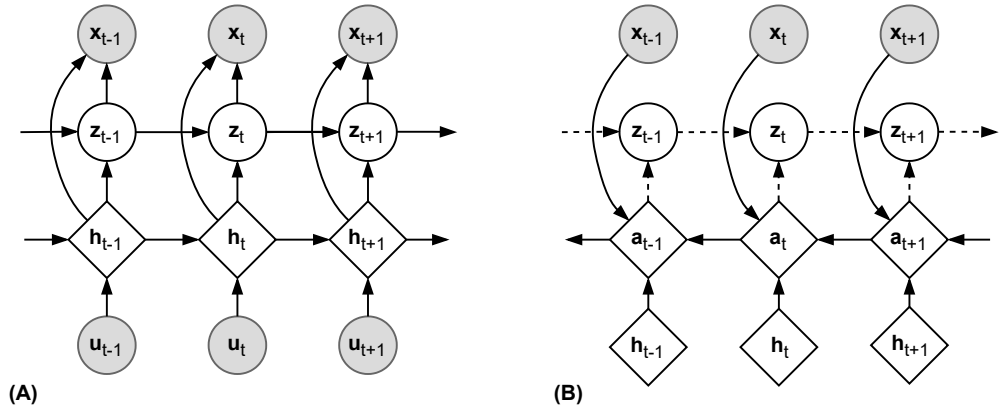


Figure 3.4: Left: the generating network of the SRNN. Right: the inference network with smoothing.

encodes information from the entire time series. This will ensure a smooth convergence of the KL-divergence, as the posterior have all of the information of the time series. This form of regularisation of the posterior will be referred to as *smoothing*.

Looking at Equation 3.4 and from Fraccaro et al. [3] the ELBO loss is given by

$$\begin{aligned}\mathcal{F}_i(\theta,\phi) = \sum_t \mathbb{E}_{q^*_\phi(\mathbf{z}_{t-1})}\Big[\mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{h}_{t:T},\mathbf{x}_{t:T})}\left[\log p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right)\right] + \\ - \mathrm{KL}\left(q_\phi\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_{t:T}, \mathbf{x}_{t:T}\right) \| p_\theta\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t\right)\right)\Big].\end{aligned} \tag{3.31}$$

Both the generative and inference model factorise over time steps, hence the ELBO above separates to a sum over time steps. Furthermore, $q^*_\phi(\mathbf{z}_{t-1})$ is the marginal posterior of $\mathbf{z}_{t-1}$ in the variational approximation to the posterior $q_\phi(\mathbf{z}_{1:t-1} \mid \mathbf{h}_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_0)$. The proof is omitted for Equation 3.31, but can be seen in Fraccaro et al. [3].

3.6 Tricks of the trade

In this section a few tricks of the trade will be introduced to handle optimisation issues. There are various approaches to handle different issues. To avoid drafting an exhaustive list of training optimisation techniques, only techniques that were found beneficial or experimented with for this thesis is introduced. The tricks used for VAEs are also often used and found beneficial for sequential deep latent models as the theory is closely related to each other.

A common issue with DLVMs is that the stochastic optimisation with an unweighted lower bound gets stuck in an undesirable stable equilibrium [26, 27]. This is often seen

by a persistent small KL value in the ELBO. The small KL value corresponds to that the prior approximates the variational posterior, $p(\mathbf{z}) \approx p(\mathbf{z}|\mathbf{x})$, and essentially means that the variational posterior carries no information. As the ELBO is maximised and the KL is minimised due to the subtraction sign in front of the non-negative KL divergence, it is sometimes beneficial to decrease the importance of minimising the KL divergence. The subtle change increases the importance of improving the rather weak likelihood term, $p(\mathbf{x}|\mathbf{z})$, at the start of the training. To do this Bowman et. al. [26] proposes using a weighting constant, β, in front of the KL term that is slowly annealed from 0 to 1 during training. Another method introduced by Kingma et al. [28] is to use *free bits*. Using free bits fixes the minimum value of the KL divergence, this modification makes sure that it is not advantageous to decrease the KL term below a fixed threshold and ensures that a minimum number of bits of information is encoded in the latent variables.

In Fraccaro et al. [3] they observe that it is challenging for a time-varying prior to keep track of the changes in the posterior distribution. This is due to the non-stationary and time-varying behaviour of the prior. To get a better variational approximation they suggest to only learn the residual between the prior and the posterior. This is done by utilising $\lim_{KLD(q,p)\to 0} q(\mathbf{z}_t) \approx p(\mathbf{z}_t)$ and letting the variational prior do one-step predictions from the $\mathbf{x}_{t-1}$ encoded latent posterior. Then as a consequence the prior mean can be approximated as [3]:

$$\widehat{\boldsymbol{\mu}}_t^{(p)} = \int \mathrm{NN}_1^{(p)}\left(\mathbf{z}_{t-1}, \mathbf{h}_t\right) p\left(\mathbf{z}_{t-1} \mid \mathbf{x}_{1:T}\right) \mathrm{d}\mathbf{z}_{t-1} \approx \int \mathrm{NN}_1^{(p)}\left(\mathbf{z}_{t-1}, \mathbf{h}_t\right) q_\phi\left(\mathbf{z}_{t-1}\right) \mathrm{d}\mathbf{z}_{t-1}, \tag{3.32}$$

where $\mathrm{NN}_1^{(p)}\left(\mathbf{z}_{t-1}, \mathbf{h}_t\right)$ is a deep neural network used to find the parameterization of the prior.

To further help the posterior, the residual mean of the prior distribution $\boldsymbol{\mu}_t^{(p)}$ and the variational posterior $\boldsymbol{\mu}_t^{(q)}$ is found at each time step t

$$\boldsymbol{\mu}_t^{(q)} = \widehat{\boldsymbol{\mu}}_t^{(p)} + \boldsymbol{\mu}_{\Delta t}^{(q)}, \tag{3.33}$$

where for the SRNN model $\boldsymbol{\mu}_{\Delta t}^{(q)} = \mathrm{NN}_1^{(q)}\left(\mathbf{z}_{t-1}, \mathbf{a}_t\right)$. This form of regularisation is known as residual q or RES_q.

The trick of letting the variational prior be directly conditioned on the posterior of the last time step, as in the RES_q-algorithm, has some slight drawbacks. The variational prior is learning directly from the variational posterior. It implies that the variational prior for the SRNN $p\left(\mathbf{z}_t \mid \mathbf{h}_t, \mathbf{z}_{t-1}\right)$ is then directly connected to the variational posterior, $q\left(\mathbf{z}_t \mid \mathbf{h}_t, \mathbf{x}_t, \mathbf{z}_{t-1}\right)$. This means when training, the gradients of the prior, are never chained together in a series over the prior, $p\left(\mathbf{z}_t \mid \mathbf{h}_t, \mathbf{z}_{t-1}\right)$. The prior is never conditioned on the previous prior sample, $\mathbf{z}_{t-1}$, when training the model. This causes the latent features of the prior to have a slight deviation from the latent features of

the posterior. The implication of this deviation is that when predicting using only the prior, the model might not be able to handle the small irregularities introduced by itself.

A possible solution is to let the prior handle its own previous sample, this method is called *overshooting* [29]. It is a tool to regularise the model based on its predictions. The most intuitive way of overshooting is to let the prior be conditioned on generated predictions, this is called *observational overshooting*. As the associated cost of generating predictions is very model dependent it is often not feasible to do observational overshooting when generating video data. Applying *latent overshooting* allows reduction of some of the computational costs associated with observational overshooting while maintaining the benefits of overshooting.

Latent overshooting is when the variational prior overshoots in latent space. Such that the overshooting is only over the variational posterior and prior optimised over the latent space. Thus the KL term of the ELBO in Equation 3.31 becomes:

$$-\frac{1}{D}\sum_{d=1}^{D}\beta_d \mathbb{E}_{\mathbf{z}_{t-1},\mathbf{z}_{t-d}}\left[\mathrm{D}_{\mathrm{KL}}\left(q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{h}_t, \mathbf{z}_{t-1}\right) \| p_\theta\left(\mathbf{z}_t \mid \mathbf{h}_t, \mathbf{z}_{t-1}\right)\right)\right], \tag{3.34}$$

here $\mathbf{z}_{t-1} \sim p_\theta\left(\mathbf{z}_{t-1} \mid \mathbf{z}_{t-d}, \mathbf{h}_{t-1}\right)$ and $\mathbf{z}_{t-d} \sim q_\phi\left(\mathbf{z}_{t-d} \mid \mathbf{x}_{t-d}, \mathbf{h}_{t-d}\right)$. β_d is a weighting for the overshoot. The parameter, β_d, is set depending on whether the model is valuing short or long term predictions. For $D = 1$, the training is done with no overshooting. For $D = 2$ the prior will be conditioned on one generated latent variable $\mathbf{z}$ from the prior.

Another problem with DLVMs occurs if the deterministic part of the model is powerful enough to explain most of the structure in the data. As there is a deterministic path from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$ this might imply that a powerful model learns to not use the stochastic part. This especially occurs for very expressive decoders, such as deep normalising flows [30, 31]. This is often seen when the KL term and the variance in the distribution of the latent variables are very small. To possibly prevent this problem from occurring the variance is often fixed or lower bounded by a specified value. This ensures that the model cannot disregard stochasticity completely, but also reduces the flexibility of the model. Another approach could be to use tricks mentioned earlier for avoiding a small KL term.

Finally, an issue with DLVMs is that the generative and/or inference model is not flexible enough. One of the symptoms from the lack of a flexible model is blurriness of the generative model as Kingma et al. [16] mentions. A solution to this is using exact log-likelihood methods in the output probability such as *Normalising Flows*, that will be introduced later.

3.7 Estimating the conditional likelihood

In practice, estimating the conditional likelihood $p(\mathbf{x}|\mathbf{z})$ is non-trivial as for some data domain, the data can be very multi-modal or discrete. Image data is quantized discrete data. This needs to be taken into account, if an exact measure of the ELBO loss is desired, such as finding the bits-per-pixel. With the bits-per-pixel (BBP) being defined as

$$b(\mathbf{x}) = -\frac{\log p(\mathbf{x}|\mathbf{z})}{D \log(2)}, \tag{3.35}$$

where D is the dimensions of an image $\mathbf{x}$ and $p(\mathbf{x}|\mathbf{z})$ the conditional likelihood. The problem arises when the conditional likelihood of the model is assumed continuous. It is impossible to estimate exact bits-per-pixel as it takes "infinite bits" to encode real numbers with infinite precision. Different methods can be applied to take the quantization of images into account. A simple solution to this problem is to use cross-entropy that is defined by the entropy between two distributions:

$$H = -\sum_{i \in \mathcal{I}} p(x_i) \log_2 p(x_i|z_i), \tag{3.36}$$

where $\mathcal{I}$ is the set of pixels and $p(x_i)$ being the true probability of pixel i and is always 1 at the index of the true bit otherwise 0. Bits-per-pixel is then found by taking the average over the set of pixels in Equation 3.36. To get the estimate of $p(x_i|z_i)$ one can use a soft-max function. From the entropy and the fact that $\frac{\log(x)}{\log(2)} = \log_2(x)$ the BPP can be estimated by

$$\begin{aligned} b(\mathbf{x}) &= -\frac{\log p(\mathbf{x}|\mathbf{z})}{D \log(2)} \\ &= -\frac{1}{D} \sum_{i \in \mathcal{I}} \log_2 p(x_i|z_i). \end{aligned} \tag{3.37}$$

In PixelCNN [32], a probability density is found over the discrete quantization of every pixel in the image. That is, for a RGB 32×32 8-bit image the probability density with dimensions $32 \times 32 \times 3 \times 256$ is assigned a 256-way soft-max function, where 256 is every discrete pixel value. The method of PixelCNN for density estimation is found both computational expensive and difficult to train. Furthermore, representing more than 256 categories, such as in RGB images gives 256^3 categories when jointly modelling the pixels. This causes memory issues when mapping the neural network activations to logits [33].

To solve the mentioned problem a discretized mixture of logistics (DMOL) distribution can be used, as described in Salimans et al. [33]. To avoid modelling the quantization levels of the pixels, in PixelCNN++ [33], the density over the quantization levels were modelled with a continuous probability and the probability is discretized when log-likelihood is evaluated over the pixel values. From Salimans et

al. [33] this can be done with a Gaussian mixture model over a logistic space, calling it a discretized logistic mixture

$$\nu \sim \sum_{i=1}^{K} \pi_i \operatorname{logistic}\left(\mu_i, s_i\right). \tag{3.38}$$

When finding the log-likelihood the probability was discretized by segmenting the continuous probability into sections by

$$p(\mathbf{x} \mid \pi, \mu, s) = \sum_{i=1}^{K} \pi_i \left[\sigma\left(\left(\mathbf{x} + 0.5 - \mu_i\right)/s_i\right) - \sigma\left(\left(\mathbf{x} - 0.5 - \mu_i\right)/s_i\right)\right], \tag{3.39}$$

where $\sigma()$ is a logistic sigmoid function. Performing such density estimation over the discrete pixel values of the conditional output probability led to faster convergence and a less memory intensive model. The conditional output could also have been modelled with a normalising flow (Chapter 4), which allows for modelling a continuous conditional output probability using bijective transformations.

CHAPTER 4

Normalizing Flows

In the previous chapter the approximated probabilistic VAE model was introduced. In this chapter another method within DLVM will be introduced: *normalizing flows*. The normalising flow model serves the same purpose of modelling a latent distribution with respect to an input distribution as the VAE model does. However, whereas VAEs do this by approximating a likelihood, the flow-based models are able to provide an exact likelihood. An exact likelihood $p(\mathbf{y})$ is preferable as the flow-based models can be trained with maximum likelihood estimation, such that it is possible to directly model the underlying variability of the data. Not only does flow-based models have exact likelihood, the flexibility of the method is also able to transform any latent distribution into an arbitrary multi-modal output distribution. This makes the flow-based model, able to model complex latent distributions and is therefore very suited for variational inference.

This chapter serves as an introduction to the general theory of normalising flows with the first sections introducing the basic theory. Section 4.3 will cover how to construct and design unconditional flow-based models. Section 4.6 will introduce conditional flow-based models. The chapter will mainly be based on the contemporary and general theory introduced by Papamakarios et al. [34] and Kobyzev et al. [35].

4.1 Definition and basics

The main idea of the normalising flows method is to transform a simple known distribution into a more complex unknown *target distribution* by some bijective mapping. This bijective mapping can be compositions of simple[1] and smaller bijective mappings. With the appropriate composition of mappings, ultimately, any known distribution can be transformed into any other distribution. This transformation of the normalising flows is eloquently described in Papamakarios et al. [34] as:

"Normalising flows operate by pushing an initial density through a series of transformations to produce a richer, more multi-modal distribution like a fluid flowing through a set of tubes."

[1] The transformations do not need to be simple, but it is often convenient, as it simplifies the problem statement and makes it possible to utilise a more modular approach to structure a normalising flow model.

The general pipeline of a normalising flow is as follows; from an initial simpler distribution, which is often called the *base distribution*, $p_u(\mathbf{u})$, a sample $\mathbf{u}$ is drawn as follows:

$$\mathbf{u} \sim p_u(\mathbf{u}). \tag{4.1}$$

Here the base distribution can be modelled with any distribution. Then with a bijective mapping $\mathbf{y}$ is expressed by a transformation of $\mathbf{u}$:

$$\mathbf{y} = T(\mathbf{u}). \tag{4.2}$$

Here the flow-based method seeks for the distribution of $\mathbf{y}$ given by $p_y(\mathbf{y}, \boldsymbol{\theta})$, to match the target distribution of the data $p_y^*(\mathbf{y})$, by optimising over the parameters $\boldsymbol{\theta}$ characterising the flow-based distribution. A defining feature of normalising flows is that the bijective transformation T must be a diffeomorphism to make the distribution of $\mathbf{y}$ well-defined. This implies the transformation T has to be invertible, and both T and T^{-1} must be differentiable.

From these conditions it is possible to obtain the target distribution over $\mathbf{y}$ with a change of variable in the base distribution:

$$p_y(\mathbf{y}) = p_u\left(\mathbf{u}\right) \left|\det\left(J_T(\mathbf{u})\right)\right|^{-1}. \tag{4.3}$$

When $\mathbf{u} = T^{-1}(\mathbf{y})$, this can also be expressed in terms of T^{-1}

$$p_{\mathbf{y}}(\mathbf{y}) = p_{\mathrm{u}}\left(T^{-1}(\mathbf{y})\right) \left|\det J_{T^{-1}}(\mathbf{y})\right|. \tag{4.4}$$

Where the Jacobian matrix J_T is the partial differentials of the transformation with respect to u_j for $j \in 1, 2, ..., D$:

$$J_T(\mathbf{u}) = \begin{bmatrix} \frac{\partial T_1}{\partial u_1} & \cdots & \frac{\partial T_1}{\partial u_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_D}{\partial u_1} & \cdots & \frac{\partial T_D}{\partial u_D} \end{bmatrix}. \tag{4.5}$$

Intuitively the transformation T is expanding or contracting the manifold-space such that the density $p_u(\mathbf{u})$ maps into $p_y(\mathbf{y})$. The absolute of the Jacobian determinant is a quantity of the relative change in volume in a small area around $\mathbf{u}$ by T. Such as if an infinitesimally small domain $\mathbf{du}$ around $\mathbf{u}$ is mapped to an infinitesimally domain $\mathbf{dy}$ around $\mathbf{y} = T(\mathbf{u})$, then $|\det J_T(\mathbf{u})|$ is equal to the volume of $\mathbf{dy}$ divided by $\mathbf{du}$. The probability mass in $\mathbf{dy}$ must be equal to the probability mass in $\mathbf{du}$. In other words, if the density of $\mathbf{y}$ is smaller than the density at $\mathbf{u}$, then $\mathbf{du}$ is expanded, and if larger, then $\mathbf{du}$ is contracted.

From Equation 4.3 with the help of T, it is possible to have a bijective mapping from the base distribution to the target distribution, but still the question remains of how to choose a suitable transformation, T. Luckily, as long as T adheres to its constraint

of being a diffeomorphism, T is *composable*. This means that the transformation can consist of more than one transformation. For a T with two composite transformations, the inverse transformation will have the following property,

$$T^{-1} = (T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1}. \tag{4.6}$$

Likewise, the determinant of the composite functions can be split up

$$\det J_{T_2 \circ T_1}(\mathbf{u}) = \det J_{T_2}\left(T_1(\mathbf{u})\right) \cdot \det J_{T_1}(\mathbf{u}). \tag{4.7}$$

With the just mentioned properties a composition of simpler transformations can transform the simple base distribution to a more complex target distribution by sequentially applying simpler transformations to the base distribution.

$$\mathbf{y} = T(\mathbf{u}) \tag{4.8}$$
$$= T_K \circ \ldots \circ T_2 \circ T_1(\mathbf{u}). \tag{4.9}$$

Equation 4.1 together with Equation 4.2 are generally speaking referred to as sampling from the model and Equation 4.3 to as evaluating the density of the model (See Figure 4.1). Evaluating the density of the model is required to find the optimisation criterion and is explained further in the next section.

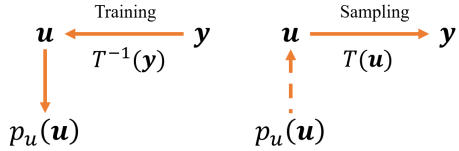


Figure 4.1: The difference between sampling and training. The solid line represent a deterministic mapping and the dashed a stochastic mapping.

These sequential transformations in Equation 4.8 will be referred to as a *flow*, as each component of the composite function slightly reversibly change the data, making $\mathbf{u}$ flow from the base distribution to the target distribution. Finally, the base distribution is commonly set as a normal distribution, whereas the naming of normalising flows arises. As the flow $T^{-1}(\mathbf{y})$ is mapping $\mathbf{y}$ to a normal distribution, i.e. in some sense normalising $\mathbf{y}$. The $T^{-1}(\mathbf{y})$ direction of the flow is generally called the *normalising direction* and $T(\mathbf{u})$ the *generating direction*.

The components of the flow T_k are referred to as *flow-layers* or *sub-flows*. The choice and design of these flow-layers will later be shown to have a trade-off between either sampling and evaluation of the model, this will be described further in Section 4.3.

4.2 Modelling and inference

To fit the target distribution of the data, $p_y^*(\mathbf{y})$ to the approximated flow-based distribution, $p_y(\mathbf{y}, \boldsymbol{\theta})$ as previously seen in Chapter 3, the fitting can be done with a measure of their discrepancy. Minimising the discrepancy measure is done over the parameters of $\boldsymbol{\theta} = \{\boldsymbol{\phi}, \boldsymbol{\psi}\}$, where ϕ is the parameters of T, and ψ of the density $p_u(\boldsymbol{u})$. Often these parameters are modelled with a deep neural network. Here the most common choice of this discrepancy measure is the KL divergence

$$\begin{aligned}\mathcal{L}(\theta) &= D_{\mathrm{KL}}\left[p_{\mathbf{y}}^*(\mathbf{y}) \| p_{\mathbf{y}}(\mathbf{y}; \boldsymbol{\theta})\right] \\ &= -\mathbb{E}_{p_{\mathbf{y}}^*(\mathbf{y})}\left[\log p_{\mathbf{y}}(\mathbf{y}; \boldsymbol{\theta})\right] + \text{const} \\ &= -\mathbb{E}_{p_{\mathbf{y}}^*(\mathbf{y})}\left[\log p_{\mathbf{u}}\left(T^{-1}(\mathbf{y}; \boldsymbol{\phi}); \boldsymbol{\psi}\right) + \log\left|\det J_{T^{-1}}(\boldsymbol{y}; \boldsymbol{\phi})\right|\right] + \text{const}.\end{aligned} \tag{4.10}$$

The measure of the distributions discrepancy equating to a expression over T^{-1} and the determinant of T^{-1} from Equation 4.4. By sampling over the true distribution of the data, $\mathbf{y}_n \sim p_y^*(\mathbf{y})$, a Monte Carlo approximation of Equation 4.10 can be found by

$$\mathcal{L}(\boldsymbol{\theta}) \approx -\frac{1}{N}\sum_{n=1}^{N} \log p_{\mathbf{u}}\left(T^{-1}\left(\mathbf{y}_n; \boldsymbol{\phi}\right); \boldsymbol{\psi}\right) + \log\left|\det J_{T^{-1}}\left(\mathbf{y}_n; \boldsymbol{\phi}\right)\right| + \text{ const}, \tag{4.11}$$

where N is the number of samples. By minimising the KL divergence, the approximate distribution $p_y(\mathbf{y}, \boldsymbol{\theta})$ converge to resemble the target distribution $p_y^*(\mathbf{y})$ as much as possible.

In practice fitting this measure is often done with stochastic based gradient methods. An unbiased estimate of the gradients of the model is given by

$$\nabla_\phi \mathcal{L}(\boldsymbol{\theta}) \approx -\frac{1}{N}\sum_{n=1}^{N} \nabla_\phi \log p_{\mathbf{u}}\left(T^{-1}\left(\mathbf{y}_n; \boldsymbol{\phi}\right); \boldsymbol{\psi}\right) + \nabla_\phi \log\left|\det J_{T^{-1}}\left(\mathbf{y}_n; \boldsymbol{\phi}\right)\right|. \tag{4.12}$$

$$\nabla_\psi \mathcal{L}(\boldsymbol{\theta}) \approx -\frac{1}{N}\sum_{n=1}^{N} \nabla_\psi \log p_{\mathbf{u}}\left(T^{-1}\left(\mathbf{y}_n; \boldsymbol{\phi}\right); \boldsymbol{\psi}\right). \tag{4.13}$$

So to be able to fit a model with SGD T^{-1}, $\det J_{T^{-1}}$ and $p(\mathbf{u}, \psi)$ has to be computed as well as their gradients. Calculating a determinant is a costly operation for which reason tricks can be implemented to restrain the computational cost of determinant operation. A selection of cost-reducing tricks are introduced in the following sections.

4.3 Composition of flows

With the composable nature of the normalising flow-based models (described in Section 4.1) a flow-based model can be composed using a series of transformations

$$\mathbf{u}_k = T_k\left(\mathbf{u}_{k-1}\right), \quad \text{for } k = 1 : K \tag{4.14}$$

where $\mathbf{u}_0$ and $\mathbf{u}_K$ is $\mathbf{u}$ and $\mathbf{y}$, respectively. The purpose of a composite flow is to approach the problem in a modular manner, such that one is able to compose the flow with a series of simpler transformations without limiting the expressiveness of the model. The inverse operator of the composed flow can be derived as

$$\mathbf{u}_{k-1} = T_k^{-1}\left(\mathbf{u}_k\right), \quad \text{for } k = K : 1. \tag{4.15}$$

The modular structure of the flow gives rise to efficient implementation, and a more tractable and simpler problem statement. With the composed flow, the $\log|\det J_T(\mathbf{u})|$ term of Equation 4.10, will be split up between each element in the flow

$$\log|\det J_T(\mathbf{u})| = \log\left|\prod_{k=1}^{K} \det J_{T_k}\left(\mathbf{u}_{k-1}\right)\right| = \sum_{k=1}^{K} \log\left|\det J_{T_k}\left(\mathbf{u}_{k-1}\right)\right|. \tag{4.16}$$

It can be intractable to calculate the determinant as it has a computational cost growth of $\mathcal{O}\left(D^3\right)$, with D being the dimension size of the data. The modular structure makes it possible to compose a normalising flow, with a series of sub-flows specially designed to have a feasible computational cost of the Jacobian determinant, and a tractable inverse. Making the problem statement more modular, and eases the implementation of the algorithm.

In theory, a composite flow can be utilised to transform the input distribution to any other distribution. A fortunate property of this form for composition is the computational cost of increasing the 'depth' of the flow, i.e. increasing the number of the transformations in the sequential composition only scales with $\mathcal{O}\left(K\right)$. Basically, by increasing the depth of the flow the expressiveness of the flow will increase with a computational cost, which scales linearly with the number of composable transformations, K.

Given the flow consists of a composite of sub-flows, the structure of the sub-flows will have a large effect in a computational trade-off between evaluation and sampling of the model. The sub-flows (T_k) will be parameterized with the ϕ-parameter. To keep these sub-flows tractable and the evaluation and sampling trade-off negligible, some practical considerations will be needed and described in this section.

The general structure of image-based flow-models is seen in Figure 4.2, here a series of flow layers is repeated K-times. The structure consists of a normalisation layer, an autoregressive layer, and a permutation layer, each layer serving a different purpose, which now briefly will be introduced.

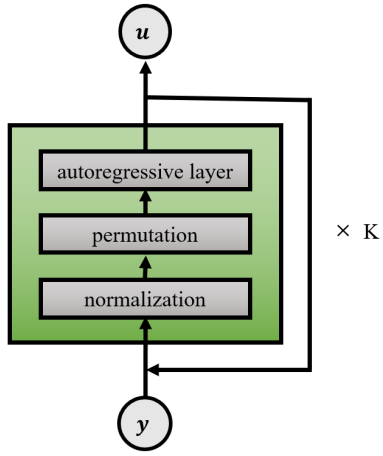


Figure 4.2: General structure of the flow-based model. The green box symbolising a flow-step. One flow-step consists of a normalisation, an autoregressive and a permutation layer.

The function of the normalisation layer is to ensure a stable convergence by making a co-variate shift of the activation. Different types of normalisation have been proposed such as batch normalisation and activation normalisation [6, 36].

The autoregressive-layer serves the purpose of applying a transformation to the data and is the layer, which is responsible for the transformative effect of the flow.

The permutation-step is done to ensure maximum mixing of the activation within the network, which effectively helps the autoregressive layer. Here different approaches have been proposed such as a binary permutation and a general permutation scheme using invertible 1×1 convolutions [6, 36].

All of these types of flow layers will be further explained in the following section. The input of the flow layer will be called $\mathbf{y}$ and the output will be $\mathbf{y}'$. All operations will be in the normalising direction and the inverse of the operation will be in the generating direction.

4.3.1 Autoregressive flow layers

The fully autoregressive layer and the coupling layer are two of the most commonly used autoregressive flow layers. The autoregressive flow layer has a recurrent structure and is modelled sequentially. The recurrent nature of the autoregressive flows inhibit sampling and the ability to parallelise the computation. A coupling layers is, in contrast to a fully autoregressive layer, the least possible autoregressive form of a

flow. Therefore, it is more efficient at sampling and easier to parallelise.

The fully autoregressive flows are a set of functions where the layer of the flow $T_{(n,\theta)}$ has the structure:

$$y'_i = \tau(y_i, \mathbf{h}_i) \qquad \text{where} \qquad \mathbf{h}_i = c_i(y_{<i}). \tag{4.17}$$

Here τ is the *transformer* and c_i the i-th *conditioner*. The transformer is a strictly monotonic function and, therefore, invertible. It is parameterized by the parameter $\mathbf{h}_i$. The transformer models how the flow is perturbed from $\mathbf{y}$ to $\mathbf{y}'$. The conditioner is modelling the parameters of the transformer and can in turn change the behaviour of the transformer. The i-th conditioner does not need to be bijective, but it cannot be dependent on the i-th step, and can only rely on previous entries less than i.

As the conditioner does not need to be bijective, the conditioner can be modelled by highly non-linear and non-invertible models, i.e. deep neural networks. As the transformer is a monotonic function, and the conditioner is non-dependent on the i-th step, it is easily invertible:

$$y_i = \tau^{-1}(y'_i, \mathbf{h}_i) \qquad \text{where} \qquad \mathbf{h}_i = c(y_{<i}). \tag{4.18}$$

Differently for forward computation, the inverse computation, all of the $y_{<i}$-steps have to be computed before y_i. This can make sampling quite computational expensive for the model. Though, some sequential flow models as IAF [28] have alleviated the computational sampling issue by inverting the flow.

The Jacobian determinant of this form of flow is clearly triangular as y'_i does not depend on $y_{>i}$, hence the Jacobian determinant will become

$$J_{f_\phi}(\mathbf{y}) = \begin{bmatrix} \frac{\partial \tau}{\partial y_1}(y_1; \boldsymbol{h}_1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{L}(\mathbf{y}) & & \frac{\partial \tau}{\partial y_D}(y_D; \boldsymbol{h}_D) \end{bmatrix}, \tag{4.19}$$

where $\mathbf{L}(\mathbf{y})$ is the lower triangle of the matrix. A property of a triangular Jacobian is that the determinant will be a sum of the diagonal elements of the Jacobian, thus the log determinant will be

$$\log|\det J_T(\boldsymbol{y})| = \log\left|\prod_{i=1}^{D} \frac{\partial \tau}{\partial x_i}(y_i; \boldsymbol{h}_i)\right| = \sum_{i=1}^{D} \log\left|\frac{\partial \tau}{\partial x_i}(y_i; \boldsymbol{h}_i)\right|. \tag{4.20}$$

A triangular matrix is very computational advantageous as it scales down the computational cost of computing the determinant from $\mathcal{O}\left(D^3\right)$ to $\mathcal{O}\left(D\right)$, where D is the dimension size of $\mathbf{y}$. This down-scaling of the computational cost is very coveted as it is possible to train a normalising flow on more and larger data.

Autogressive flows are *universal approximators* given that the transformer and conditioner are flexible enough to represent any function arbitrarily well [34]. A simple transformer for this type of autoregressive flow model could be an *affine transformer*:

$$y_i' = \tau\left(y_i; \boldsymbol{h}_i\right) = \alpha_i y_i + \beta_i \quad \text{where} \quad \boldsymbol{h}_i = \{\alpha_i, \beta_i\}. \tag{4.21}$$

The affine transformer is in its most basic nature just a linear scaling of the dependant variables, whereas the parameters for this scaling is parameterized by the conditioner. As any other flow layer the affine transformation also needs to be invertible. If $\boldsymbol{\alpha} = 0$, the function can not be inverted, since the term with y_i in Equation 4.21 would be lost. To alleviate the problem of $\boldsymbol{\alpha} = 0$, an exponential function is used $\boldsymbol{\alpha} = \exp(\tilde{\boldsymbol{\alpha}})$. For the affine transformer the expressiveness is limited, given that the base distribution is often parameterized with a Gaussian distribution. Sequentially applying affine transformers on a Gaussian distribution will only lead to other shifted and scaled Gaussian distribution expressions $p_{\mathrm{y}'}\left(\mathrm{y}_i' \mid \mathrm{y}_{<i}'\right)$ [34].

The log determinant of an affine transformer is defined by

$$\log\left|\det J_{f_\phi}(\mathbf{y})\right| = \sum_{i=1}^{D} \log\left|\alpha_i\right| = \sum_{i=1}^{D} \tilde{\alpha}_i. \tag{4.22}$$

The affine autoregressive flow transformer have α and β as parameters, which depend on the conditioner. A simple choice for this form of conditioner network could be using a Recurrent Neural Network (RNN), which was introduced in Section 3.3.

A way to keep the sequential transformation of the autoregressive flow to its bare minimum is by *coupling layers*. Coupling layers will maintain $\mathcal{O}\left(\frac{D}{2}\right)$ computational cost of the determinant, while being a non-sequential calculation and thus easily parallelised.

The main idea of coupling layers is to split up the data entry at index d, whereas $y_{i<d}$ is conditioned on the rest of the data. A schematic of an affine coupling is seen in Figure 4.3. Commonly d is set to $d = \frac{D}{2}$, such that half of the data is conditioned on the other half

$$\begin{aligned} \mathbf{y}_{\leq d}' &= \mathbf{y}_{\leq d} \\ (h_{d+1}, \ldots, h_D) &= c\left(\mathbf{y}_{\leq d}\right) \\ \mathrm{y}_i' &= \tau\left(\mathrm{y}_i; h_i\right) \ \text{ for } i > d. \end{aligned} \tag{4.23}$$

The conditioner c will be parameterized as a neural network, as this ensures high non-linearity and gives shape to very complex transformations. With the inverse of Equation 4.23 being

$$\begin{aligned} \mathbf{y}_{\leq d} &= \mathbf{y}_{\leq d}' \\ (h_{d+1}, \ldots, h_D) &= c\left(\mathbf{y}_{\leq d}\right) \\ y_i &= \tau^{-1}\left(y_i'; h_i\right) \ \text{ for } i > d \end{aligned} \tag{4.24}$$

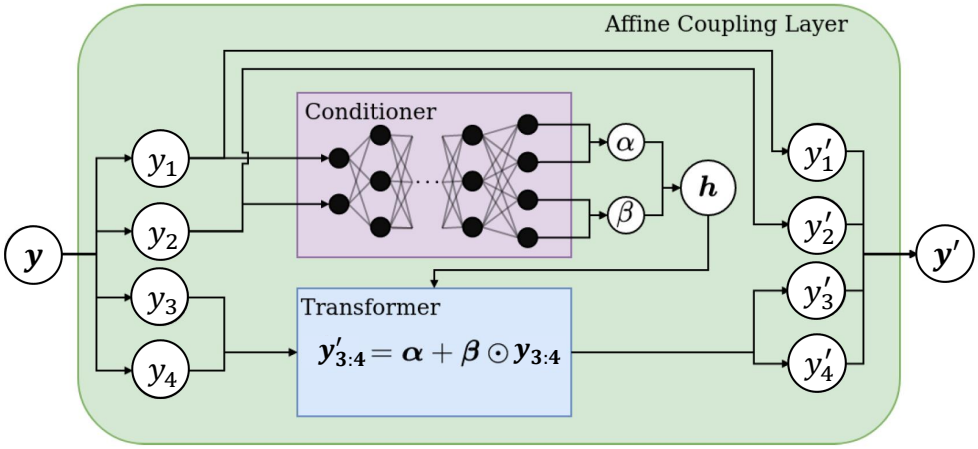


Figure 4.3: Schematic of the affine coupling layer with a input $\boldsymbol{y}$ of size 4 (Tygesen, [37]).

A coupling layer with an affine transformer is an *affine coupling layer* and is defined as:

$$\tau\left(y_i; \boldsymbol{h}_i\right) = \alpha_i y_i + \beta_i \quad \text{where} \quad \boldsymbol{h}_i = \{\alpha_i, \beta_i\}\,. \tag{4.25}$$

As only half of the data is perturbed in one affine coupling layer step, a composition of sequential applied affine coupling layers will perturb only one half of the activations. To alleviate this problem an alternating coupling layer scheme can be utilised where the half of the data, which is perturbed, is alternated between the affine coupling layers. Another way to alleviate the problem for data mixing is by using different types of invertible data permutations. When data permutation is inserted between affine coupling layers, it increases the effectiveness of the coupling layers. These permutation schemes are further elaborated on later in Section 4.3.2.

The affine coupling layer will only perturb half of the data, the Jacobian for the affine coupling layer is

$$J_T(\mathbf{y}) = \begin{bmatrix} \mathbf{I} & 0 \\ L(\mathbf{y}) & \frac{\partial \tau}{\partial \mathbf{x}_{>d}}\left(\mathbf{y}_{>d}; \mathbf{h}_{>d}\right) \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ L(\mathbf{y}) & \boldsymbol{\alpha}\mathbf{I} \end{bmatrix}. \tag{4.26}$$

$L(\mathbf{y})$ is the lower triangle of the matrix and can be ignored as the determinant of a triangular matrix is the sum of the diagonal elements. As a result

$$\log\left|\det J_T(\mathbf{y})\right| = \log\left|\prod_{i=1}^{D} \frac{\partial \tau}{\partial y_i}\left(y_i; \mathbf{h}_i\right)\right| = \sum_{i=d}^{D} \log\left|\frac{\partial \tau}{\partial y_i}\left(y_i; \mathbf{h}_i\right)\right| = \sum_{i=d}^{D} \log\left|\alpha_i\right|, \tag{4.27}$$

which reduces the cost of the determinant $\mathcal{O}\left(D^3\right)$ to $\mathcal{O}\left(\frac{D}{2}\right)$ [34]. This shows coupling layers are a very efficient method of transforming the data in a cost efficient manner with respect to the determinant. Clearly affine coupling layers have many advantages, but they come with a trade-off. With the increased efficiency of the coupling layer, a single coupling layer is not able to express any autoregressive behaviour, regardless of the expressiveness of a deep neural net in the conditioner. As a result, an autoregressive flow with a single coupling layer is no longer an universal approximation, as only half of the data is transformed in a single affine coupling step. Nevertheless, it is possible to create expressive flows by combining multiple coupling layers with interlinking permutation layers, giving the flow a chance to transform all of the data.

Affine coupling layers are a common used type for autoregressive flow layers in image based flow models. As this thesis main focus is to generate sequential image based data, the rest of the sections in this chapter will mainly focus on affine coupling layers and its inter-relating effects on the other flow layers.

4.3.2 Permutation

Given the modular approach of the construction of normalising flows, as described earlier in this section, several different types of flow-layers can be interconnected to increase the performance of the flow. One of these modules of the flow, is the permutation layer, see Figure 4.2. A permutation layer will increase the performance of the flow model. The idea is to permute the data in such a manner, that all of the data is thoroughly mixed. The mixing ensures a greater effect of the affine coupling layers, as coupling layers only perturb one-half of the data. To permute the data several different methods can be used. The two methods focused on in this section is the masked coupling layer and the generalised permutation method of linear flow layers.

Masked affine coupling is a simple approach for making permutation of the data set by making masked partitions of the affine coupling layers.

$$\mathbf{y}' = b \odot \mathbf{y} + (1 - b) \odot (\mathbf{y} \odot \exp(s(b \odot \mathbf{y})) + t(b \odot \mathbf{y})) \tag{4.28}$$

When first introduced b was a binary mapping of either a masking scheme across channels or a checkerboard mapping across height and width of the pixels [36], see Figure 4.4. The permutation of the binary mask is applied in an alternating pattern between the coupling layers. This ensures an appropriate mixing of activations through the flow, increasing the effectiveness of the coupling layer.

Linear flow layers are a generalisation of the masked perturbation scheme. Linear flow layers perturb across all data points by applying a matrix-vector product. This ensures a mixing over all of the data points in the flow

$$\mathbf{y}' = \mathbf{W}\mathbf{y}. \tag{4.29}$$

Its inverse is

$$\mathbf{y} = \mathbf{W}^{-1}\mathbf{y}'. \tag{4.30}$$

There are some problems associated with this kind of flow layer. One being, that a linear flow layer is a linear problem for the invertible transformation. If the solution for a linear problem is not non-singular, then the transformation is non-bijective. Another problem for this type of flow is $\mathbf{W}$ will scale $\mathcal{O}\left(D^2\right)$ with the data, and $\mathcal{O}\left(D^3\right)$ for the inversion of $\mathbf{W}$, this is infeasible for high-dimensional data set as the computational cost would exceed the performance gain. To avoid the problem of non-singular inversion, different restrictions can be applied to $\mathbf{W}$. These restrictions can be that $\mathbf{W}$ is a binary matrix with only one entry of 1's in each row and column. This ensures the invertibility of the matrix.

In Kingma et. al. [6] a linear flow perturbation scheme was introduced as a 1×1 convolution over the channels in an image, i.e. for an image of size $[c, h, w]$, the transformation is

$$\mathbf{y'}_{i,j} = \mathbf{W}\mathbf{y}_{i,j} \quad \text{inversely: } \mathbf{y}_{i,j} = \mathbf{W}^{-1}\mathbf{y'}_{i,j}, \tag{4.31}$$

where every element $\mathbf{y}_{i,j}$ having a size of $[1, c]$ and $\mathbf{W}$ a size of $[c, c]$. The initial weights for the matrix is set by a QR-factorization, which eases invertibility. The permutation matrix $\mathbf{W}$ is trainable, as the weights is assumed to not deviate from the initialisation in the non-singular domain. This type of linear flow over the channels ensures that the computational cost only scales with the amount of channels.

Later in Section 4.4 more efficient structural designs of the network architecture for image-based flow models will be discussed. These structural designs will increase the performance of the network, but significantly scale the number of channels. Even with this increase of channels the 1×1-convolution is a feasible permutation method of large scale network training.

To further increase the performance of the linear flow a PLU-decomposition can be applied to the permutation matrix $\mathbf{W}$. The PLU-decomposition decreases the computational cost of the inversion of the matrix in Equation 4.31 from $\mathcal{O}\left(D^3\right)$ to $\mathcal{O}\left(D^2\right)$ and the computational cost of the determinant from $\mathcal{O}\left(D^2\right)$ to $\mathcal{O}\left(D\right)$. With a PLU-decomposition the permutation matrix is decompositioned into 3 different matrices, described as

$$\mathbf{W} = \mathbf{PLU} \tag{4.32}$$

Here $\mathbf{P}$ is the permutation, and $\mathbf{L}$, $\mathbf{U}$ is the lower and upper triangular. To adhere to the invertibility of the linear flow procedure, $\mathbf{L}$ and $\mathbf{U}$ are restricted to being only positive.

With a PLU-decomposition, the absolute determinant of $\mathbf{W}$ can be computed in $\mathcal{O}(D)$ time by

$$|\det \mathbf{W}| = \prod_{i=1}^{D} L_{ii} U_{ii}. \tag{4.33}$$

With a backward forward substitution the inversion of W can be found in $\mathcal{O}\left(D^2\right)$ cost. To train a network with this type of permutation $\mathbf{L}$ and $\mathbf{U}$ are free parameters, with $\mathbf{P}$ being fixed to its randomly initialised value.

4.3.3 Normalisation

Batch normalisation is a method usually used to create a more stable convergence, when training neural networks. Batch normalisation is done by reducing the co-variate shift of the networks activations [38] and is essentially a combination of two affine transformations, the first calculated by the batch statistics and the second set by free parameters. The batch statistics are found from the activations

$$\begin{aligned} \hat{\mu} &= \frac{1}{m} \sum_{i=1}^{m} y_i \\ \hat{\sigma}^2 &= \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{\mu})^2 \\ \widehat{y}_i &= \frac{y_i - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon}}. \end{aligned} \tag{4.34}$$

Subtracting the mean and dividing by the variance of the activations, it is effectively normalised. To then restore the representation power of the network a second affine transformation is applied

$$y_i' = \alpha \widehat{y}_i + \beta \equiv \mathrm{BN}\left(y_i\right), \tag{4.35}$$

where α, β is learned parameters. The invertible transformation for batch normalisation is then

$$\mathrm{BN}^{-1}\left(y'\right) = \hat{\mu} + \frac{y' - \beta}{\alpha} \odot \sqrt{\hat{\sigma}^2 + \epsilon} = y. \tag{4.36}$$

As the batch normalisation operation consists of two affine transformations, the Jacobian will be a triangular matrix as thus the determinant is easily calculated by

$$\det J_{\mathrm{BN}}(\mathrm{y}) = \prod_{i=1}^{D} \frac{\alpha_i}{\sqrt{\hat{\sigma}_i^2 + \epsilon_i}}. \tag{4.37}$$

Batch normalising ensures smoother training gradients and leads to higher stability when training a network. However, the variance of the network activations noise, introduced by batch normalisation, is inversely proportional to the size of the mini-batch. Hence, a different approach is needed when a large batch size cannot be

achieved, e.g. when a large batch size utilisation will lead to a memory limit on the GPU. One of these alternatives is *Activation normalisation* (ActNorm), introduced in Kingma et al. [6]. ActNorm is designed to alleviate the memory usage problem. It is an affine transformation introducing a learnable scale and bias parameter, which performs the same co-variate shift as the batch normalisation, but it is batch independent

$$\mathbf{y'}_{i,j} = \boldsymbol{\alpha}\mathbf{y}_{i,j} + \boldsymbol{\beta}. \tag{4.38}$$

As a layer in normalising flow it is also reversible:

$$\mathbf{y}_{i,j} = \frac{\mathbf{y'}_{i,j} - \boldsymbol{\beta}}{\boldsymbol{\alpha}}. \tag{4.39}$$

The function is differentiable and the Jacobian determinant will be

$$\det J(\mathrm{y}) = \prod_{i=1}^{D} \alpha_i. \tag{4.40}$$

4.4 Multi-scale architecture

Flow-based models have been used as probabilistic multi-modal image generating models by applying the above stated methods. Given the computational intensiveness of flow layers and flow models applied to images, steps have been taken to decrease computational cost. One such step is in Dinh et al. [36], a multi-scale structure is proposed. By using the multi-scale architecture approach, they adopt the method from the likes of other image generating models by making the flow model more latent and "deeper" [39]. To do this two different operations are introduced, a *squeeze*-operator and a *split* operator.

The squeeze operator takes and reduces the spatial dimensions, while increasing the channels, making the number of elements remain the same. So given an image of size $\{c, h, w\}$ the squeeze operator transforms the image to dimensions $\{4c, \frac{h}{2}, \frac{w}{2}\}$. The operation is done in such a way that the mixing of the channels is maximised, see Figure 4.4. This operation is clearly invertible.

The main idea of the split operator is to be able to remove some of the activations within the entirety of the network, by replacing a subset of the activations with an approximation. The split operator does this by first given a set of activations $\mathbf{u}$, the activations is split into two halves $\mathbf{u}_0$ and $\mathbf{u}_1$. With this split, a conditional distribution of $\mathbf{u}_1$ is found by conditioning on $\mathbf{u}_0$, and estimating the distribution parameters e.g. if the conditional distribution is a Gaussian distribution $p(\mathbf{u}_1 \mid \mathbf{u}_0) = \mathcal{N}\left(\mathbf{u}_1; \mu(\mathbf{u}_0), \sigma^2(\mathbf{u}_0)\right)$ where $\mu(\cdot)$, $\sigma^2(\cdot)$ are estimated by a deep neural network. This conditional Gaussian distribution is referred to as a *conditional split prior* or *conditional prior*.

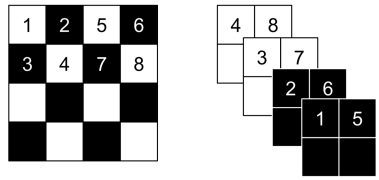


Figure 4.4: A schematic of the masking and squeezing operator. To the left a spatial checkerboard masking scheme is applied to an $1 \times 4 \times 4$ tensor. To the right the reduced $4 \times 2 \times 2$ tensor with a channel masking (Dinh et al. [36]).

To invert this form of operation, a sample of the conditional prior is drawn:

$$\mathbf{u}_1 \sim p(\mathbf{u}_1 \mid \mathbf{u}_0). \tag{4.41}$$

As the conditional distribution depends on a neural network and is sampled from a normal distribution, it follows that this split operation will be approximately invertible, compared to other flow-based methods which are completely invertible. The invertibility of the split operation will depend highly on the strength of the neural network used to approximate the distribution. So by implementing the split operation there will be a trade-off between losing complete invertibility and a reduction in data size. Even with this kind of trade-off, it is advantageous to implement this type of operation periodically in a flow network as it reduces the number of elements in the model while also maintaining the complexity and flexibility of normalising flows [7, 36]. In Dinh et al. [36], they also argue that the granular insertion of $\mathbf{u}$, also help with the optimisation by distributing the objective throughout the flow. The contribution to the loss function will be the log probability of the conditional distribution, $\log p(\mathbf{u}_1 \mid \mathbf{u}_0)$.

Periodically applying the squeeze and split methods lead to the multi-scale architecture depicted in Figure 4.5 (A). Here the multi-scale architecture consists of L levels each consisting of the operations: squeeze, K *flow steps*, and one split operation. The flow steps being a series of flow layers, see Section 4.5. Using the multi-scale architecture the choice of L decides the depth of the model, and K the complexity and number of parameters for the model. A deeper normalising flow will be more expressive [40], but also more computational costly. With a multi-scale structure the conditional priors of the split layers will have the factorization:

$$p_u(\mathbf{u}) = \prod_{l=1}^{L} p_u(\mathbf{u}_l), \tag{4.42}$$

where $p_u(\mathbf{u}_l)$ represents the conditional split prior Equation 4.41 of the split-operator in the l-th layer of the L blocks of the multi-scale architecture. Effectively this can

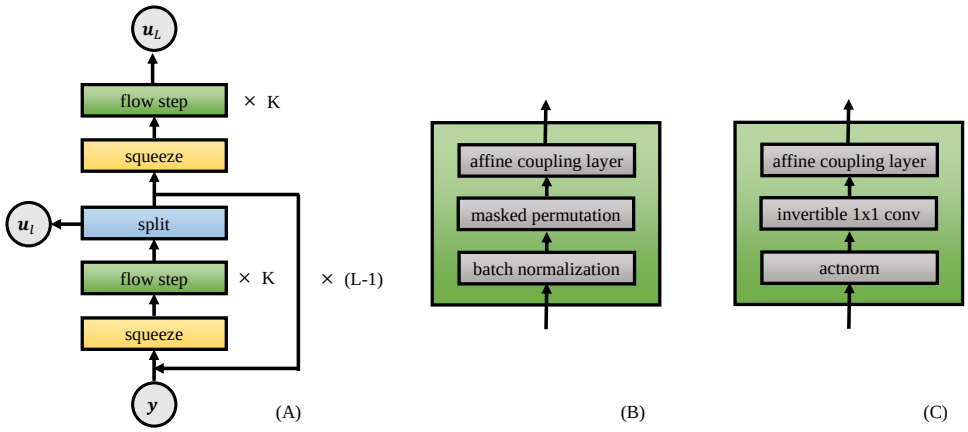


Figure 4.5: Normalising multi-scale flow structure (A), periodically apply the squeeze and the split. (B) the flow step of the RealNVP model. (C) the flow step of the Glow model.

be seen as a granular division of the base distribution in stages, each conditional split prior approximating the features of the flow at different levels.

The contribution to the loss in Equation 4.10 will be a contribution from the multi-scale structure in the form of the conditional priors and Jacobian of the flow-layers [1, 34].

4.5 Types of Flow Models

Two predominant flow-models have emerged for image generation: real-valued non-volume preserving (RealNVP) [36] and Glow [6]. With Glow being an extension of RealNVP that eliminates some of the problems introduced by the RealNVP algorithm. RealNVP utilises a multi-scale structure together with a flow-step (see Figure 4.5) consisting of batch normalisation, a masked permutation, and an affine coupling layer. This model was the first model to perform sharp visually compelling full-colour image generation with a flow based method [34].

The Glow model [6] took the RealNVP model and improved aspects of the model such as memory efficiency and generalisation of the permutation. Glow introduced the linear flow to generalise the permutation step, and the activation normalisation,

as a more memory efficient and less noisy normalisation step compared to the batch normalization used in RealNVP. A flow step in Glow consists of an activation normalisation layer, a linear flow layer, and an affine coupling layer. The linear flow layer acting as a general permutation of the data, such as to maximise the performance of the affine coupling layer as described in Section 4.3.2.

4.6 Conditional normalising flows

In the previous sections the probability density $p_u(\mathbf{u})$ was found given the dependent variable $\mathbf{u}$. This probability density can be conditioned, so that the probability density now depends on external covariates $\mathbf{x}$. Then the probability density becomes conditional $p(\mathbf{u}|\mathbf{x})$. Conditioning on an external variable will give some control over the probability density, as the probability distribution is now conditioned on the external covariates. These types of conditional distributions can be utilised in a lot of different aspects such as super resolution of the images [41], sequential problems (e.g. video generation) [1], and image content transfer [41]. This section will shortly present ways to incorporate conditioning into the already established flow-based model, also known as conditional normalising flows (CNF).

The general approach of incorporating the conditioning variable $\mathbf{x}$ is to condition it on every flow-step in the normalising flow, the base distibution, and the conditional split prior. Then the log-likelihood is

$$\log p_\theta(\boldsymbol{u} \mid \boldsymbol{x}) = \log p_u(\boldsymbol{u} \mid \boldsymbol{x}) - \sum_{i=1}^{K} \log \left| \det \boldsymbol{J}_{T^{(k)}_{\boldsymbol{x},\boldsymbol{\theta}}}(\boldsymbol{u}_{k-1}) \right|, \tag{4.43}$$

where $T^{(k)}_{\boldsymbol{x},\boldsymbol{\theta}}$ is the k-th conditioned flow-step. The conditioned flow-step, deviates from the normal flow-steps, by now also being conditioned on external covariates. While still sampling and training of the model will be conceptually the same as the unconditional normalising flow, see Figure 4.6.

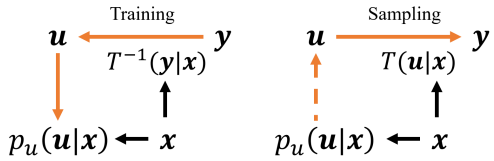


Figure 4.6: Conceptual schematic of the sampling and the training process for a CNF, with the external covariates $\mathbf{x}$.

Given a flow-based architecture for image generation as previously described in Section 4.5, the way to condition these models is described in the following paragraphs.

The flow-step consists of a normalisation layer, a permutation layer and an affine coupling layer, with only the affine coupling layer being conditional [7]. The normalisation layer is unconditional as the covariate shift is invariant of the condition, such that the activation reaching the normalisation layer is still indirectly conditioned by the conditional affine coupling layer. The permutating layers' main purpose is to increase the effectiveness of the affine coupling layer by mixing the activations within the network. This is also true for the conditional flow model, so the effectiveness of the flow-step does not increase by directly conditioning on the permutation step. To summarise: it is only the affine coupling layer of the conditioned flow-step, which are directly affected by the conditioning.

Previously in Section 4.3.1, the affine coupling method had the conditioner and the transformer, which got the dependent variables transformative values α and β from the conditioner. To extend this model with a condition, the external covariates are inserted into the conditioner, so that α and β will use the context of the external data in the transformer.

$$\begin{aligned} y'_{\leq d} &= y_{\leq d} \\ (h_{d+1}, \ldots, h_D) &= c\left(y_{\leq d}, x\right) \\ y'_i &= \tau\left(y_i; h_i\right) \text{ for } i > d \end{aligned} \tag{4.44}$$

All of the conditional affine coupling layers together with a representation of the conditional network can be seen in Figure 4.7. To extract features and possibly reduce the dimensionality of the external covariates, a deep neural network can be used. In Figure 4.7 the extracting network is depicted as a feed forward neural network, but it can be any type of neural network. This network is called the conditioning network. It is assumed that the conditioning network is extracting high-level features of the data, such that if more than one flow-step is in the model the same features from the conditioning network can be utilised for other flow-steps. This is done since it is assumed the extracted features have a rich embedding of higher order features, which can partly be extruded in the different affine coupling steps. By only extracting the features once, the computational cost becomes more scalable.

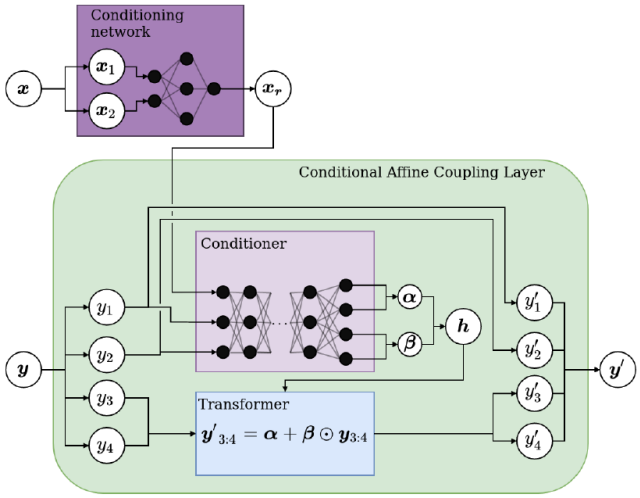


Figure 4.7: A schematic of the conditional affine coupling layer, with an input $\mathbf{y}$ and a condition $\mathbf{x}$ (Tygesen, [37])

As seen from Figure 4.6, the conditional base distribution can be conditioned in the following way:

$$p_u(\boldsymbol{u} \mid \boldsymbol{x}) = \mathcal{N}\left(\mathbf{u}; \mu\left(\mathbf{x}\right), \sigma^2\left(\mathbf{x}\right)\right), \tag{4.45}$$

where $\mu(\cdot)$, $\sigma^2(\cdot)$ are estimated by deep neural networks. By conditioning on the base distribution of the flow, it is now possible to have a more flexible base distribution which can adapt to the external covariates and lead to a better flow distribution.

Combining the previously mentioned conditioned flow-layers together with a multi-scale architecture, it is possible to create a conditioned model as seen in Figure 4.8. Here the structural design of the conditional normalising flow model only slightly deviates form the unconditioned multi-scale model (Section 4.4). The conditional prior of the periodically split-operator, will now also be conditioned on the external covariates. Conditioning the conditional split prior on the external covariates ensures the approximate conditioned distribution, will have the context of the external covariates to give a better representation of the data. The conditional split prior is defined as:

$$p\left(\mathbf{u}_1 \mid \mathbf{u}_0, \mathbf{x}\right) = \mathcal{N}\left(\mathbf{u}_1; \mu\left(\mathbf{u}_0, \mathbf{x}\right), \sigma^2\left(\mathbf{u}_0, \mathbf{x}\right)\right). \tag{4.46}$$

The multi-scale factorization will be:

$$p_u\left(\mathbf{u} \mid \mathbf{x}\right) = \prod_{l=1}^{L} p_u\left(\mathbf{u}_l \mid \mathbf{x}\right), \tag{4.47}$$

where $p_u(\mathbf{u}_l \mid \mathbf{x})$ represents the conditional split prior in Equation 4.46 of the split-operator in the l-th layer for the L blocks of the multi-scale architecture. The contribution to the loss will be $\log(p_u(\mathbf{u} \mid \mathbf{x}))$.

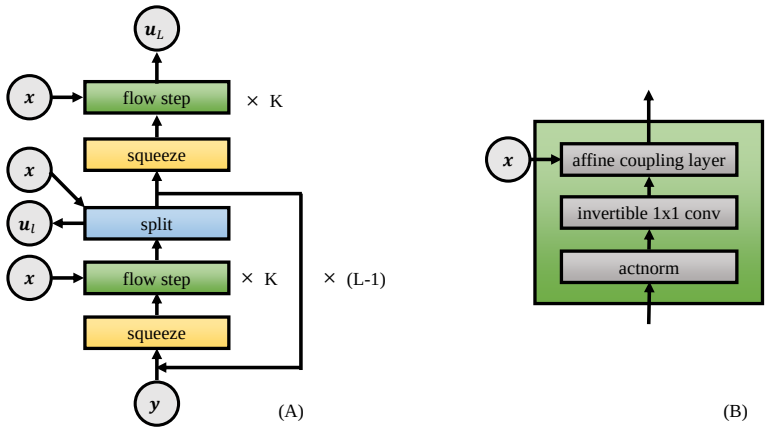


Figure 4.8: (A) a schematic of the conditional normalising flow multi-scale architecture where the split operator and the flow step is conditional. (B) The conditional flow step is seen. Of the three flow layers within the flow step, only the affine coupling layer is conditional.

CHAPTER 5

Recurrent Flow Network

Following, the theory of the Recurrent Flow Network (RFN) model is presented. Gammelli et al. [5] first introduced RFN. Described in Layman's terms, the model unifies a slightly modified version of SRNN and a conditional normalising flow. It models highly complex data distributions and its variability, such as seen in spatio-temporal data. As the algorithms introduced in Chapter 3, the RFN consists of a generative and an inference model. Figure 5.1 illustrates the probabilistic graphical model.

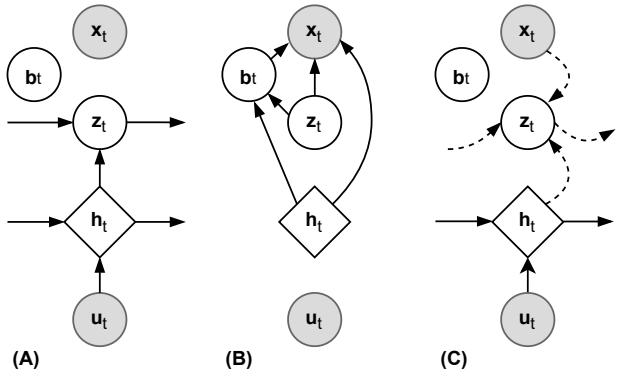


Figure 5.1: PGM of the RFN-model. (A) is the transition function. (B) is the emission function. (C) is the inference network.

The model is composed of three parts, a transition, an emission, and the inference. These terms relate to state-space theory. This thesis will not go into depth with the naming of the different parts, but refers to Fraccaro et al. [2] for thorough explanation. Briefly stated, the transition and emission together define the generative model. The transition will update the latent variables and the deterministic hidden states based on their previous values and some condition variable $\mathbf{u}_t$. The emission applies the conditional normalising flow to reconstruct $\mathbf{x}_t$ based on latent variables from the

conditional base distribution of the flow and the updated values of the transition function. Finally, inference is done by an inference model inspired by a SRNN [3]. The RFN model builds on many of the same principles and theory described in Chapter 3 and Chapter 4. To avoid redundancy some theory of the RFN will be presented without further explanation.

5.1 Generative Model

In Section 3.5 the SRNN was mentioned. Like the SRNN model, the RFN model has a deterministic and a stochastic temporal part, but with an added separation of states for the spatial variability [5]. The deterministic temporal part models the overall structure of the data. The temporal stochastic dynamics, i.e the statistical uncertainty such as randomness over a time sequence, will be modelled by the variational prior and posterior.

The first step for the generating part of the model is the transition. This is where the information of the sequential data is represented by a neural network in such a way that the emission is able to reconstruct the image from the high-level features extracted in the transition. The transition function consists of a fully deterministic recurrent network $t_{\theta_\mathbf{h}}$ parameterized by $\theta_\mathbf{h}$ that acts as the memory of the model and finds a sequential representation of the extracted features. This is done to determine the deterministic hidden states $\mathbf{h}_t$ by

$$\mathbf{h}_t = t_{\theta_\mathbf{h}}\left(\mathbf{h}_{t-1}, \varphi_\tau^{\text{ext}}\left(\mathbf{u}_t\right)\right), \tag{5.1}$$

where $\varphi_\tau^{\text{ext}}$ is a feature extractor extracting high-level information of the external covariates $\mathbf{u}_t$ over time step t. As previously stated, in this thesis $\mathbf{u}_t = \mathbf{x}_{t-1}$.

The stochastic part of the sequential data is encoded by sequentially applying a transformation on the latent variables $\mathbf{z}_t$, which is sampled from the distribution of the conditional prior. In this case a Gaussian distribution is parameterised to sample $\mathbf{z}_t$:

$$\mathbf{z}_t \sim \mathcal{N}(\boldsymbol{\mu}_{0,t}, \text{diag}(\boldsymbol{\sigma}_{0,t}^2)), \quad \text{where } \left[\boldsymbol{\mu}_{0,t}, \boldsymbol{\sigma}_{0,t}\right] = t_{\theta_\mathbf{z}}\left(\mathbf{z}_{t-1}, \mathbf{h}_t\right), \tag{5.2}$$

where $t_{\theta_\mathbf{z}}$ is a deep neural network that estimates the parameters $\boldsymbol{\mu}_{0,t}$ and $\boldsymbol{\sigma}_{0,t}$. With the two parameters $\mathbf{h}_t$ and $\mathbf{z}_t$ encoding the information of the data the emission function is prepared.

The emission function is able to use the rich embedding of the latent space $(\mathbf{h}_t, \mathbf{z}_t)$ to generate a reconstruction of the data adhering to the probabilistic constraints[1] of the problem. Unlike the SRNN, the decoder part of the model is defined by a CNF

[1] For video generation the probabilistic constraints will be the domain of plausible outcomes.

as described in Section 4.6.

The conditional base distribution of the flow, $p(\mathbf{b}_t \mid \mathbf{z}_t, \mathbf{h}_t)$, uses the learned representations $(\mathbf{h}_t, \mathbf{z}_t)$, such that the conditional flow is able to act on the sequential conditional properties from the transition

$$\text{Conditional Prior: } \mathbf{b}_t \sim \mathcal{N}(\boldsymbol{\mu}_{b,t}, \text{diag}(\boldsymbol{\sigma}_{b,t}^2)), \quad \text{where } \left[\boldsymbol{\mu}_{b,t}, \boldsymbol{\sigma}_{b,t}\right] = f_\psi(\mathbf{z}_t, \mathbf{h}_t), \tag{5.3}$$

where the latent flow samples are $\mathbf{b}_t$. These samples are sampled from a Gaussian distribution with mean and standard deviation $\boldsymbol{\mu}_{b,t}, \boldsymbol{\sigma}_{b,t}$ estimated by some deep neural network, $f_\psi(\mathbf{z}_t, \mathbf{h}_t)$. The affine coupling of the flow is likewise conditional on the learned representations of the sequential data:

$$\begin{aligned} \text{Conditional Coupling: } \mathbf{b}_{t,d+1:D} &= \mathbf{x}_{t,d+1:D} \odot \exp(s_\psi(\mathbf{x}_{t,1:d}, \mathbf{z}_t, \mathbf{h}_t)) + t_\psi(\mathbf{x}_{t,1:d}, \mathbf{z}_t, \mathbf{h}_t) \\ \mathbf{b}_{t,1:d} &= \mathbf{x}_{t,1:d}, \end{aligned} \tag{5.4}$$

where s_ψ and t_ψ are parameterized by deep neural networks. Using the above affine coupling the CNF will be able to generate predictions and reconstruct $\mathbf{x}_t$.

5.2 Inference Model

The variational approximation of the posterior, $q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t)$, for the RFN depends on $\mathbf{z}_{t-1}$, $\mathbf{h}_t$ and $\mathbf{x}_t$. This dependence makes sure the variational approximation contains information from previous time steps through the deterministic and stochastic representations, creating an implicit dependence on $\mathbf{h}_{1:t}$ and $\mathbf{x}_{1:t}$ through $\mathbf{z}_t$. The latent variables given $\mathbf{x}_t$ is then found by

$$\mathbf{z}_t \mid \mathbf{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_{z,t}, \text{diag}(\boldsymbol{\sigma}_{z,t}^2)), \quad \text{where } \left[\boldsymbol{\mu}_{z,t}, \boldsymbol{\sigma}_{z,t}\right] = \varphi_\tau^{\text{enc}}(\mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t), \tag{5.5}$$

where $\varphi_\tau^{\text{enc}}$ is parameterised by an encoder network defining the parameters of the approximate posterior distribution, $\boldsymbol{\mu}_{z,t}$ and $\boldsymbol{\sigma}_{z,t}$. The approximate posterior is chained together with the other states through the hidden state of the RNN $\mathbf{h}_t$, such that the variational approximation will have the following factorization:

$$q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) = \prod_{t=1}^{T} q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t). \tag{5.6}$$

5.3 Training

To learn the generative and inference model, RFN is utilising AVI. AVI allows the algorithm to jointly estimate the variational parameters θ and ϕ, which are the parameters of the generative and inference model, respectively. Again this is done by maximising the step-wise ELBO by optimizing the parameters θ and ϕ using SGD.

$$\begin{aligned}\mathcal{F}_i(\theta, \phi) = \sum_{t=1}^{T} \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x}_{1:T})} \Big[\log p_\theta \left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t \right) \Big] \\ - \mathrm{D}_{\mathrm{KL}} \left(q_\phi \left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t \right) \| p_\theta \left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t \right) \right).\end{aligned} \tag{5.7}$$

The KL divergence effectively ensures that the variational prior $p_\theta \left(\mathbf{z}_t \mid \mathbf{z}_{t-1} \mathbf{h}_t \right)$ is tracking the variational posterior $q_\phi \left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t \right)$. Furthermore, $p_\theta \left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t \right)$ defines the conditional likelihood for the reconstruction loss. A derivation of the ELBO can be seen in Appendix A.2.

CHAPTER 6

Evaluation Methods

A wide range of evaluation methods are used in video generation problems. This chapter will focus on evaluation methods that were used in this thesis, these include Mean-Sqaure Error (MSE), Structural Similarity Index Measure (SSIM), Peak Signal-to-Noise Ratio (PSNR), Learned Perceptual Image Patch Similarity (LPIPS) and Frechét Video Distance (FVD).

6.1 Metrics and Evaluation Protocols

Peak Signal-to-Noise Ratio (PSNR) is a measure very similar to the MSE, and is actually defined from it. The mean squared error is found between every pixel between the constructed image, and the ground truth.

$$MSE = \|\mathbf{x} - \mathbf{y}\|_2^2 , \tag{6.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two images. From the MSE measure the PSNR is then defined as:

$$PSNR = 10 \cdot \log_{10} \left(\frac{L_D^2}{MSE} \right) , \tag{6.2}$$

where L_D is the dynamic range of the image, defined as $(2^B - 1)$ with B being the number of bits within the image. Typically B is 8, as RGB images are 8-bit. The PSNR metric is often used to measure the quality of a reconstruction from a lossy compression codec, used in image-compression tasks. The PSNR measure is an unbound measure, where having a higher values is better. Given that the PSNR measure is based on the MSE, models which are optimising over a MSE-loss will naturally have a tendency to score a higher PSNR value. Therefore, often blurry images are scored higher. This behaviour applies to all losses and metrics, as a model trained with the same loss function as the evaluation metric will naturally try to minimise the error for that specific metric [42].

Structural similarity index measure (SSIM) [43] is likewise a similarity measure of the perceived quality between a ground truth image and a reconstructed or distorted image. SSIM is defined as follows

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)\,(2\sigma_{xy} + c_2)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)}. \tag{6.3}$$

The SSIM measure will be calculated within a $N \times N$ window of pixels. In Equation 6.3 all the following values are found for the two images, where μ, σ are denoting the mean and the standard deviation and σ_{xy} is the covariance between the images. The parameters c_1 and c_2 stabilises the denominator and are dependent on the dynamic range of the image L_D. The parameters are then set to $c_1 = (k_1 L_D)^2$, $c_2 = (k_2 L_D)$, and $(k_1, k_2) = (0.01, 0.03)$ by default. The range of SSIM is bound between $[-1, 1]$ where higher is better.

Given the window based approach of SSIM it is possible that the measure scores higher for image sequences with a low temporal dependence. Many video data sets have a static background that is not changing between frames, this will make SSIM overconfident and give it a high score. As such, SSIM is not necessarily giving an adequate evaluation of the temporal changes in the video.

Learned Perceptual Image Patch Similarity [44] is an image based similarity metric. It is found by taking the L^2 distance between extracted features from a pretrained `AlexNet` [45] evaluated on patches of each image. To find the LPIPS distance the generated images are then pairwise paired with the ground truth reference [46].

Finally, a commonly used probabilistic metric is the *Fréchet Video Distance* (FVD) introduced in 2019 [47]. It is a metric for generative models of video. It is based on the underlying principles of the *Fréchet Inception Distance* (FID) [48]. To evaluate the quality of generated images the FID is often used. FID uses the 2-Wasserstein distance to compare two Gaussian distributions, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathcal{N}(\boldsymbol{\mu}_\tau, \boldsymbol{\Sigma}_\tau)$. The FID is given by:

$$d^2((\boldsymbol{\mu}, \boldsymbol{\Sigma}), (\boldsymbol{\mu}_\tau, \boldsymbol{\Sigma}_\tau)) = \|\boldsymbol{\mu} - \boldsymbol{\mu}_\tau\|_2^2 + \mathrm{Tr}\left(\boldsymbol{\Sigma} + \boldsymbol{\Sigma}_\tau - 2\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}_\tau\right)^{1/2}\right). \tag{6.4}$$

The parameters $(\boldsymbol{\mu}, \boldsymbol{\Sigma}), (\boldsymbol{\mu}_\tau, \boldsymbol{\Sigma}_\tau)$ are obtained from the distribution over the activations of a deep layer in a powerful classification network, often a pre-trained Inception network [49]. The activations er estimated when original images and generated images are fed to the network, respectively. The Gaussian distribution is used because it has the highest entropy of all the real-valued distributions given a finite first and second order moments. This makes sure the distribution incorporates the least possible information, minimising the prior information in the distribution. This theoretically ensures the evaluation is objective rather than subjective [50]. FID gives an insight in how the distribution of the activation of deep layers in the network differ between generated and original images. The hypothesis being that, if two images have the same class, according to the network, they will exhibit similar activation in the deeper layers.

FVD extends the FID to work for sequential image data. The authors argue that the Inception network in FID only focuses on objects in images and the current visual presentation at any given point in time, but does not take into account the temporal

coherence of visual content across a sequence of frames. Furthermore, the authors also observe a potential for large error when estimating Gaussian distributions in Equation 6.4 over the learned feature space. As a solution to the mentioned problems, the paper suggests using a pre-trained Inflated 3D Convnet and a kernel-based approach using polynomial kernels, both of which will not be covered in this thesis as they are out of scope. If curious we refer to the paper for a thorough explanation, [47]. The benefit of using FVD over PSNR or SSIM is that it tries to compare the spatio-temporal distribution of the image sequences. Unlike PSNR and SSIM that only take into account individual frames FVD will take into account the whole sequence of frames. Finally, FVD is also shown to outperform PSNR and SSIM in agreeing with subjective human judgement [47].

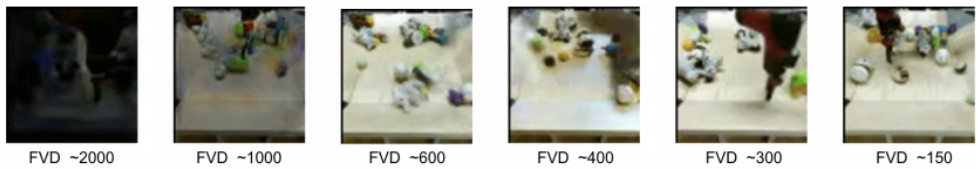


Figure 6.1: Examples of generated videos ranked according to FVD on the BAIR dataset. Lower is better. Figure taken from Unterthiner et al. [47].

Evaluation of prediction sequences between its visual appearance and the ground truth is not always reliable. Pixel-wise similarity metrics penalise all predictions that deviate from the ground truth. Given the probabilistic nature of video generation, some cases arise, where a blurry prediction nearly accommodating the ground truth will get a better score than a sharper and more plausible, but imperfect generation would. As such pixel-wise metrics can not always reflect how accurate a model is at capturing dynamics and variability [42].

With the stochastic nature of video it can be difficult to estimate how well the model performs. This is because when stochastic temporal dynamics are present, the ground truth and the prediction might not match. As such an otherwise excellent prediction, can get a terrible metric score using the conventional metrics; SSIM, PSNR and LPIPS. To overcome this issue, a simple approach is to repeatedly generate prediction sequences with the same initial conditions. If repeated enough times the stochastic prediction will ”match” the ground truth. This allows one to select the best matching generated image sequence to the ground truth reference sequence. The approach will give a more unbiased estimate of how good the model is able to mimic the underlying tendencies of the sequential data, and is the most common method for evaluating probabilistic video generating models. It is sometimes referred to as max. PNSR/SSIM or min. LPIPS [1, 42].

CHAPTER 7

Data

As the use of deep neural networks within probabilistic models can be very flexible in terms of modelling power, their behaviour might also be difficult to predict[2]. As the model used in this thesis consists of a lot of different components it is crucial to have an understanding of each component, e.g. cost function, inference model, generative model, the influence and interaction of the different blocks in the models architecture. To better understand the underlying dynamics of the model, it is necessary to apply the model. In this chapter the data sets used to evaluate the model are presented. In total 5 data sets are used to evaluate the components of the model. However, 2 of the data sets will not be included in the results as they are not deemed as relevant to the evaluation of the RFN. The excluded data sets and some few results can be seen in Appendix A.3 and Appendix A.4. Both data sets led to a deeper understanding of the individual components in the RFN, specifically the SRNN and the conditional Glow.

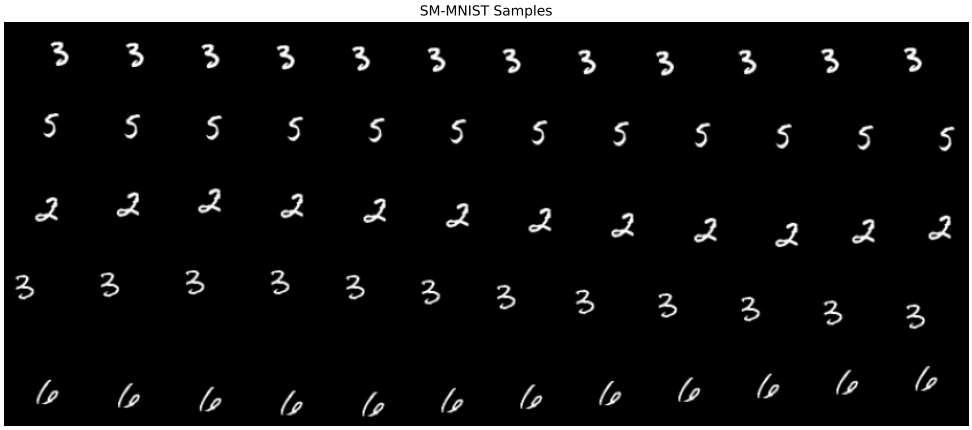


Figure 7.1: 5 sequences of single digit SM-MNIST with a velocity of 4 and 12 frames.

7.1 Stochastic Moving MNIST

Stochastic Moving MNIST (SM-MNIST) from the Modified National Institute of Standards and Technology is a data set based on the well-studied MNIST data set that consists of 70000 unique images of handwritten digits ranging from 0 to 9. The data is split up into 60000 training images and 10000 testing images[51]. Each image is 28×28 pixels with only a single grayscale channel. The 784 pixel values are 8-bit meaning the intensities of the image have the discrete range: 0 to 255. The MNIST data set is modified to make it a sequential data set. First, the background of the images are resized to 64×64, while the digit size will slightly increase to 32×32 size. This follows the literature standard [52] and will give the digit room to move while not being too small. To make the data set sequential a start position that lies within the frame is sampled from a discrete uniform distribution. An initial velocity vector for the digit is sampled between $[-4, 4]$ in both a horizontal and vertical direction. The velocity vector will determine how the position of the digit updates in each frame. The velocity vector will remain the same between frames, until the digit hits the border of the image. When the digit hits a border a new velocity vector is sampled. The sampling of the velocity vector is the stochastic part of the data set, the rest is deterministic. The number of frames in a sequence are user-specified. Furthermore, each image can contain a specified number of digits, often chosen as 1, 2 or 3 digits. In Figure 7.1 examples of samples can be seen.

Figure 7.2: 5 sequences of the BAIR data set with a sequence length of 10 frames.

7.2 BAIR action-free robot pushing

The action-free robot pushing data set is from Berkeley Artificial Intelligence Research (BAIR) [53]. It is a small robot arm pushing around small objects in a stochastic fashion and is the data set containing the most realistic images compared to the other data sets used. For convenience the data set will simply be referred to as BAIR. It contains 43520 images where 256 is used for testing and 43264 for training. The images are 64×64, with 3 channels, each channel is 8-bits. The sequence length is user specified. In Figure 7.2 it can be seen that the BAIR data is quite complex, not only does the robot arm move randomly around at each step, but sunlight, colours, shades, and different objects will also make the problem significantly more complex to model.

7.3 Recognition of human actions

The KTH (Kungliga Tekniska högskolan) data set[54] contains 6 types of human actions (walking, jogging, running, boxing, hand waving and hand clapping). There are 25 subjects performing different actions in 4 different scenarios: outdoors, outdoors with scale variation, outdoors with different clothes and indoors. The total number of video sequences is therefore $25 \cdot 4 \cdot 6 = 600$. Each sequence is 8-bit with a single colour channel. The data set is split by subjects, in this thesis 16 persons are in the training data set and 9 in the test set. The backgrounds are homogeneous and the camera is static with 25 fps. Each sequence lasts 4 seconds, so there is 100 frames in a single recording. It is possible to read more about the data set here [55]. In Figure 7.3 a single frame from different video sequences can be seen. The uncertainty lies in the appearance of subjects, actions they perform and how they perform it. The

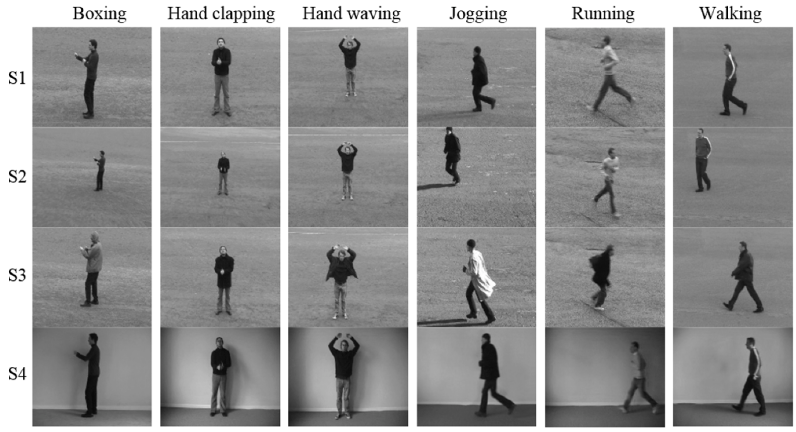


Figure 7.3: Single images [56] showing different actions, subjects and scenarios.

splitting of the data follows the same procedure as in Winkler et al. [7]. As the data set is not very large, but the sequences are long, sub-sequences are generated matching the required frames that is used for training. The test set is composed of the last 5 subjects. It is randomly sampled sub-sequences by the number of frames specified.

CHAPTER 8

Method

In this chapter the experimental approach and the architecture of the model are presented. As the RFN consists of many parts the architecture will be described in sections. In Section 8.1 references of the architectures used for the different tested models are listed. In Section 8.2 the architecture of the RFN model for video data is presented. Section 8.3 will describe the preprocessing of the data. Finally, training and evaluation will shortly be explained in Section 8.4 and Section 8.5.

8.1 Architecture of secondary models

As the VRNN and SRNN have only been applied to 2D temporal data the architecture has been modified to take into account higher dimensional data. The modifications make it feasible to compare the results of the RFN to the results of VRNN and SRNN. The architecture follows the graphical representation of Chung and Fraccaro et al. [3, 25], and can be seen in Figure 8.1. Linear neural networks are replaced by convolutional neural networks to take the spatial variability of images into account. In Figure

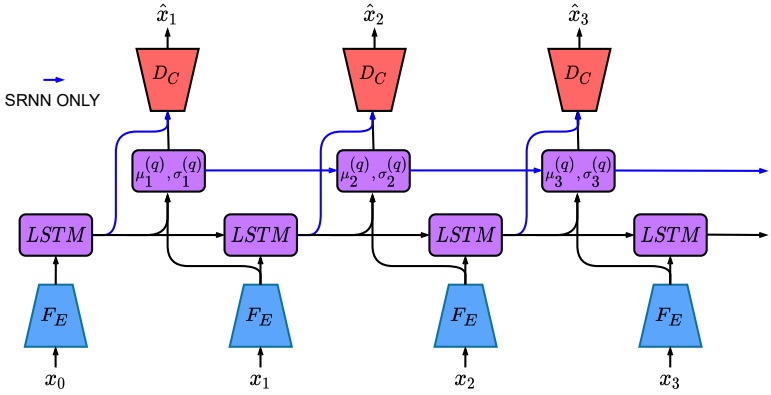


Figure 8.1: Schematic representation of inference over a sequence in VRNN and SRNN. Here the blue arrows indicate dependencies only relevant to the SRNN. F_E is the feature extractor, D_C is the decoder.

8.1 a schematic representation of the models temporal inference over a sequence is presented. The difference between the SRNN and VRNNs temporal dependencies are indicated with blue arrows. F_E is the feature extractor following a VGG structure [39]. The main purpose of the feature extractor is to extract spatial features of $\mathbf{x}_t$ to get a latent representation of the data. The VGG structure will be explained later in Section 8.2.1. The recurrent network is a ConvLSTM that takes into account the spatial dimensions of the data. ConvLSTM has shown to outperform the conventional LSTM that uses conventional linear neural networks to estimate its parameters [15].

For hyperparameter tuning, the output probability of the SRNN and VRNN is approximated by a Gaussian distribution $p(\mathbf{x}|\mathbf{z}_t, \mathbf{h}_t) \approx \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{h}_t), \nu\mathbf{I})$ with ν being a learned parameter, this form of approximation in the output probability will lead to fast convergence [33, 57]. To get a comparatively measure for the ELBO of the RFN as well as a fair comparison between models, the conditional output probability $p(\mathbf{x}|\mathbf{z}_t, \mathbf{h}_t)$ is exactly estimated for the SRNN and VRNN. This is done by using a discretized mixture of logistics introduced in the PixelCNN++ paper by Salimans et al. [33]. It allows for an exact measure of the true underlying output distribution of the 8-bit discrete video data. By using this discretization an estimate of the bits-per-pixels can be found (BPP) and compared directly to the RFN score. Modelling the conditional output probability exactly can be a difficult problem and requires more time to make the models converge. In all cases, the true posterior and prior of both models are modelled using a Gaussian distribution in the variational bottleneck.

8.2 Architecture of RFN

Conceptually the RFN is partitioned into three blocks; a feature extractor, a conditioning network, and a conditional normalising flow (CNF) network. Each partition is based on a certain block seen in the full overview of the model, Figure 8.2. Overall, the RFN is a combination of the SRNN and a conditional Glow model as explained in Chapter 5.

8.2.1 Feature extractor

The feature extractor acts as an encoder of $\mathbf{x}_t$ and the structure follows the structure of a DCGAN or VGG network. The feature extractor is designed to match the structure of the conditioning network and the base distribution of the CNF network. The feature extractor network structure will scale accordingly to the multi-scale structure of the CNF. The structure can be seen in Table 8.1. The channel growth rate after each multi-scale L-level block of convolutions is set to 2. The number of blocks depend on the multi-scale structure of the normalising flow. One block is assigned for each L-level and each block reduces the spatial size by 2. For data with 64×64 spatial dimensions, here the L level ranges from 1 to a maximum of 6. For the sixth L-level

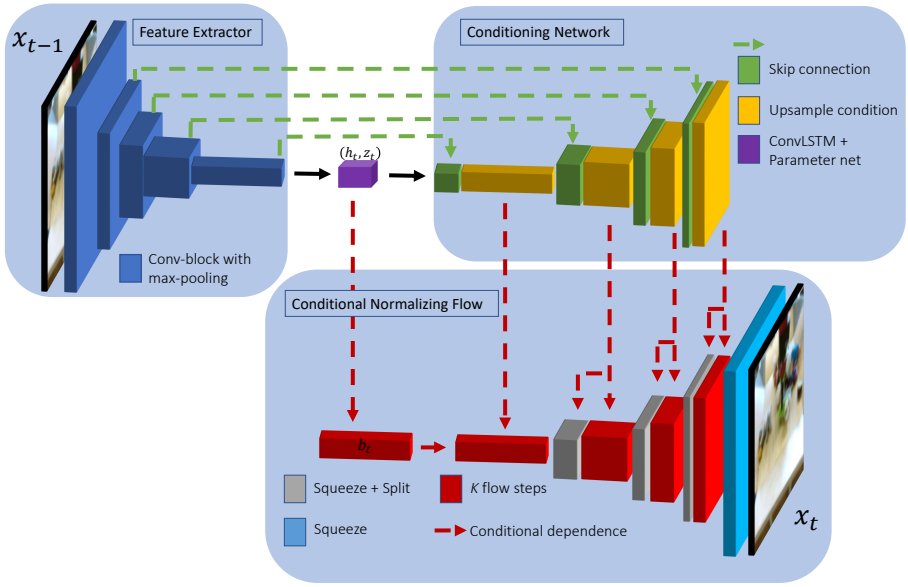


Figure 8.2: Schematic of the RFN architecture. Utilising the multi-scale architecture of the conditional flow. Either a VGG or DCGAN network [39, 58] is used to feature extract and upsample features. The bottleneck is where parameters of $\mathbf{h}_t$ and $\mathbf{z}_t$ are estimated by a ConvLSTM and a parameter network of convolution neural networks.

the spatial size is 1×1, of the output for the last block. Both the feature extractor and conditioning network use batch normalisation and leaky ReLUs with a default negative slope of 0.2 to increase stability of convergence.

8.2.2 Bottleneck

The bottleneck indicated by the purple block in Figure 8.2, is where the recurrent latent variables $\mathbf{h}_t$ and $\mathbf{z}_t$ are estimated. In Figure 8.3 the temporal dependence of the different parameters are seen. For the determination of $\mathbf{h}_t$ a ConvLSTM with kernel size of 3×3 is used instead of a regular LSTM using conventional linear neural networks, as stated before the ConvLSTM shows better performance on spatial data [15].

The parameter network is already fed with a rich representation of the data from the feature extractor. Therefore, only a single block with 2 convolution neural network layers is used to estimate the parameters $[\boldsymbol{\mu}_t^{(p)}, \boldsymbol{\sigma}_t^{(p)}]$ and $[\boldsymbol{\mu}_t^{(q)}, \boldsymbol{\sigma}_t^{(q)}]$ for the prior and posterior distribution, respectively, to generate $\mathbf{z}_t$. In general the optimal number of channels for $\mathbf{h}_t$ and $\mathbf{z}_t$ is data dependent. However, $\mathbf{h}_t$ should be higher than the

#L	Network structure	Spatial size of output
-	Input	$H \times W$
1	8, 8, Pooling	$\frac{H}{2} \times \frac{W}{2}$
2	16, 16, Pooling	$\frac{H}{4} \times \frac{W}{4}$
l	$8 \cdot 2^l$, $8 \cdot 2^l$, Pooling	$\frac{H}{2^l} \times \frac{W}{2^l}$

Table 8.1: Example of the VGG structure. L indicates the levels of the multi-scale structure in the CNF. When setting the L-level of the RFN, an associated number of convolution blocks are added to the feature extractor and conditioning network. This is shown in the network structure, each number is the channels of a convolution, with 3×3 kernel size, followed by a max-pooling at the end. DCGAN uses the same structure but replaces max-pooling with convolutions with stride 2 for down-scaling the spatial size.

channels of the stochastic latent variables, $\mathbf{z}_t$ as they are responsible for making a generalisation of the data, while $\mathbf{h}_t$ captures the spatial features and deterministic dynamics [16].

8.2.3 Conditioning Network

The conditioning network uses the same structure as the feature extractor, just reversed. It takes the latent variables $\mathbf{h}_t$ and $\mathbf{z}_t$ as input. It then scales the latent variables to match each L-level of the CNF model given by $(\mathbf{h}_t^l, \mathbf{z}_t^l)$. The conditioning network uses long skip connections from the feature extractor. Each output from a block in the feature extractor will be concatenated on the input of each block in the conditioning network. This makes the structure of the conditioning network containing a VGG network resemble the structure of a U-net [59]. The skip connections allow for fine grained details of $\mathbf{x}_t$ to be recovered. It does this by helping the model reconstruct the input, while letting the variational bottleneck focus more on the temporal dynamics[60]. The VGG version in the conditioning network uses 2d-nearest-neighbour upsampling and the DCGAN uses transposed convolutions. The conditioning network will pass spatial matching latent features on to the flow and is described in greater detail in Subsection 8.2.4.

8.2.4 Conditional normalising flow

The CNF is based on Section 4.6 and the work of Kingma and Winkler et al. [6, 7]. The Glow architecture was chosen as it makes significant improvements to the Real-NVP architecture. The improvements have proven to lead to promising results, such as in VideoFlow introduced by Kumar et al. [1]. VideoFlow implements a conditional Glow structure and applies it to video data. The CNF follows the same multi-scale structure and will be conditioned as described in Chapter 5. The CNF is conditioned

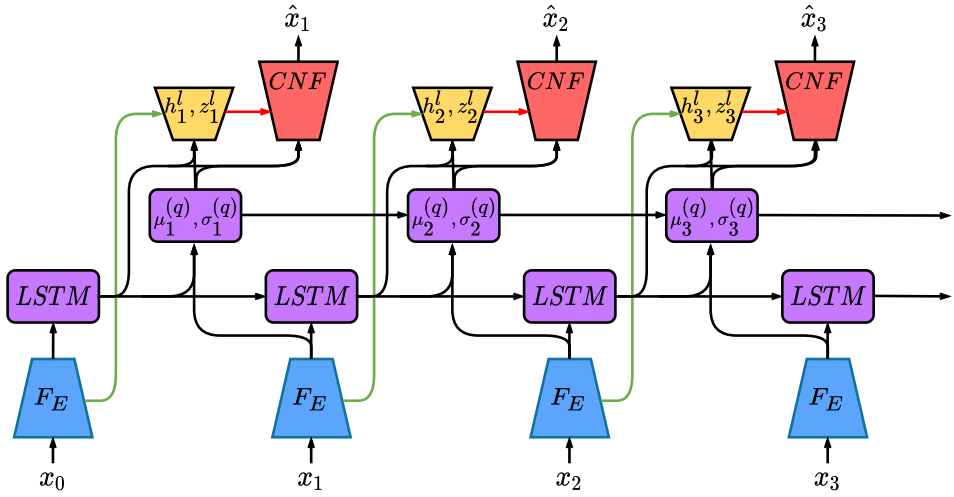


Figure 8.3: The temporal dependencies of the RFN model showing how reconstruction of $\mathbf{x}$ is accomplished. F_E is the feature extractor, CNF is the conditional normalising flow, and the LSTM block contains the ConvLSTM. The colours are matching the colours in Figure 8.2. This means that the green arrow shows the skip connection from the extractor to the conditional network and the red arrow to the CNF from the conditional network. The colour code of the boxes match that of Figure 8.2.

on the spatial matching latent features $(\mathbf{h}_t^l, \mathbf{z}_t^l)$ from the conditioning network in each L-level of the multi-scale architecture and is given as conditional input to the conditional split prior and affine coupling flow layers.

The affine coupling layer is defined with the output of the conditioner being $(\log \boldsymbol{\alpha}, \boldsymbol{\beta}) = c(\mathbf{b}_{\leq d}, \mathbf{h}_t^l, \mathbf{z}_t^l)$. It has been shown to be advantageous to restrict the parameter α with an affine clamping. The restriction is defined by $\log(\boldsymbol{\alpha})_{clamped} = s_{scale} \tanh \log(\boldsymbol{\alpha}) + s_{shift}$, with s_{scale}, s_{shift} being learned parameters. This method of clamping on the affine coupling layers is the same as in RealNVP[36], and it was also used in Glow [6]. The clamping prevents exploding gradients by limiting the parameter domain within a reasonable boundary. The conditioner network in the affine layer consists of three layers of convolutional neural networks with kernel sizes $3, 1, 3$. The two first layers are using ActNorm for regularisation and a ReLU activation. Here the bias and weights of the last convolutional layer are initialised to 0 in accordance with Kingma et al. [6]. This initialisation prevents accumulation of noise for deep levels of the flow, essentially allowing a deeper network and faster convergence [1].

The conditional split prior of the RFN is defined as

$$p\left(\mathbf{b}_1^l \mid \mathbf{b}_0^l, \mathbf{h}_t^l, \mathbf{z}_t^l\right) = \mathcal{N}\left(\mathbf{b}_1^l; \boldsymbol{\mu}\left(\mathbf{b}_0^l, \mathbf{h}_t^l, \mathbf{z}_t^l\right), \boldsymbol{\sigma}^2\left(\mathbf{b}_0^l, \mathbf{h}_t^l, \mathbf{z}_t^l\right)\right). \tag{8.1}$$

The conditional split prior is applied in the same way as previously described in Section 4.6. It is parameterised by 3 convolutional neural networks that estimate $\boldsymbol{\sigma}$ and $\boldsymbol{\mu}$. The networks follow the same initialisation and regularisation as in the affine coupling layer. Instead of taking the exponential on the log-scale neural network output we noticed that it is crucial to use a Softplus activation to get $\boldsymbol{\sigma}$ and avoid unstable gradient behaviour. The reason being that Softplus has a slower growth than the exponential function limiting spikes in $\boldsymbol{\sigma}$ that might cause instability.

All ActNorm layers are initialised by two learned parameters, a bias and a scale. The bias and the scale are initialised as the mean and $1/(\sigma_a + \epsilon)$, respectively, from the first batch of activations reaching the ActNorm layer. Here σ_a is the standard deviation of the activation and ϵ is a small constant of 10^{-6} to stabilise the denominator. Finally, PLU decomposition is used in the 1×1 invertible convolutions as described in Subsection 4.3.2 to make the training less computational demanding.

A temperature is also added to the CNF to adjust sample quality. It scales the variance of the Gaussian distributions. Earlier work on likelihood-based generative models [61] show promising results sampling using this method. To do this the standard deviation of the base distribution $p_\theta(\mathbf{b}_t|\mathbf{z_t}, \mathbf{h_t})$ and the conditional split priors are multiplied by a factor T [6].

8.3 Data preprocessing

To ensure the convergence in the loss function seen in Equation 4.10 the data is preprocessed in two ways.

Uniform binning correction: Modelling discrete data using a real-valued distribution might lead to narrow high density spikes on each of the discrete values. To avoid this instability uniform binning is implemented. Let $\mathcal{D} = \left\{\mathbf{x}^{(i)}\right\}_{i=1}^{N}$ be some form of data with n-bit discrete quantization of values from $[0, 255/256]$. To avoid having infinite densities a small uniformly sampled noise $\mathbf{u} \sim \mathcal{U}(0, \frac{1}{2^B})$ is added to the data matching its discretization level. By dequantizing the data, it is possible to avoid having infinite densities at the data points, which can lead to misbehaving optimisation of the log-likelihood. When this small amount of noise is added, we need to take into account the error it causes. As such a constant term c is added to the log-likelihood, where $c = -M \cdot \log(2^B)$ and M is the dimensionality of the data [1, 6, 36]. This is applied on every data set the RFN is trained on, as each data set is images containing discrete pixel values.

Data transformation: The data is mapped into the domain $[-0.5, 0.5]$ by doing a transformation $\frac{\mathbf{x}}{2^{\mathrm{B}}} - 0.5$ following Kingma et al. [6]. When sampling an invertible mapping is performed to map it back to the original data domain.

8.4 Training

The models are all trained with an Adam optimiser [62]. The learning rate is data dependent, but a general value around 0.0003 seems to be good in most cases. After a fixed number of training steps the learning rate is linearly annealed to zero. Annealing allows for smaller steps in the optimiser and can therefore lead to a better convergence. The KL term is scaled with an annealed factor β, which is linearly increased from $\beta \approx 0$ to 1, in accordance with Higgins et al. [63] and as described in Section 3.6. The models have also been prepared in such a way that it is possible to enable RES_q, latent overshooting or smoothing as described in Section 3.6 and Section 3.5.

All code were written by using the deep learning library `PyTorch` [64] and trained on a single Nvidia Tesla V100 GPU.

8.5 Evaluation and tuning

For hyperparameter tuning and comparing different techniques a validation set is used to prevent a bias in the test set. The validation will mainly be on the SM-MNIST data set. A validation set consisting of 1000 sequences is split from the SM-MNIST and BAIR training data. No hyperparameter tuning is done on the KTH data set. Mainly PSNR, SSIM and LPIPS will be used for fine-tuning, FVD is too expensive. To evaluate the models, all of the evaluation methods described in Chapter 6 are used on the test set.

CHAPTER 9

Experiments and results

In this chapter the results of the RFN, SRNN, and VRNN are presented. The results will be discussed in greater detail in Chapter 11. The experiments will contain both quantitative and qualitative results. All results are shown for converged models on the test set, with experiments based on the validation set. Convergence was determined by no improvement in the loss over 20 epochs. For the RFN only the final models used to evaluate the results are trained to convergence. For the RFN models used in the hyper-optimisation convergence is not fully met, but trained for a significant amount of time, this is due to time constraints. The consequences of this is discussed in Chapter 11. Each RFN is on average trained 4 days, while the SRNN and VRNN is trained until the convergence criteria is satisfied, on average this took 2 days. Specified hyper-parameters for each model can be seen in Appendix, Section A.5. A RFN model trained on each data set will be compared to state-of-the-art and other baseline generative models for video data by using the commonly used FVD [1] score. Section 9.1 will focus on evaluating the regularisation methods. In Section 9.2 hyper-parameters of the RFN are investigated. The rest of the sections will present the results when training a RFN on the data sets shown in Chapter 7.

Before introducing the results, we want to clarify the difference between a warm-up period and conditioning directly. When a warm-up is mentioned it implies that the model is recurrently updated using a number of conditions. In this case, the condition is defined as a previous ground truth frame to the current frame of a video. The warm-up period is used before a prediction is generated, and more warm-up steps in general lead to better predictions as parameters have seen more of the video sequence. Warm-up is different to conditioning directly, where the model sees a number of frames and each prediction is directly conditioned on one or several frames at each time step. This distinction coheres with current literature. The models introduced in this thesis are only able to condition on a single frame, while e.g. VideoFlow [1] can directly condition on 3 frames.

9.1 SRNN experiments

RFN is computationally demanding. As mentioned, it is unifying a SRNN and conditional Glow model. Adding temporal recursion does not make it less demanding. As such, generalisations have to be made due to time constraints. To decrease the computational load some regularisation methods will only be trained on the SRNN and then generalised to the RFN model. It is not optimal, but it is a feasible solution when considering the time constraints.

In Figure 9.1 experiments for smoothing and RES$_q$ applied to the SRNN are shown. The models are trained on 10 frames with one conditioning frame. For predicting a warm-up period of 5 frames is used when evaluating. To evaluate the models the SSIM/PSNR/LPIPS score is sampled 100 times for each time step to find the maximum (or minimum for LPIPS) score of the measure. Each model is trained until convergence on the full data set of SM-MNIST.

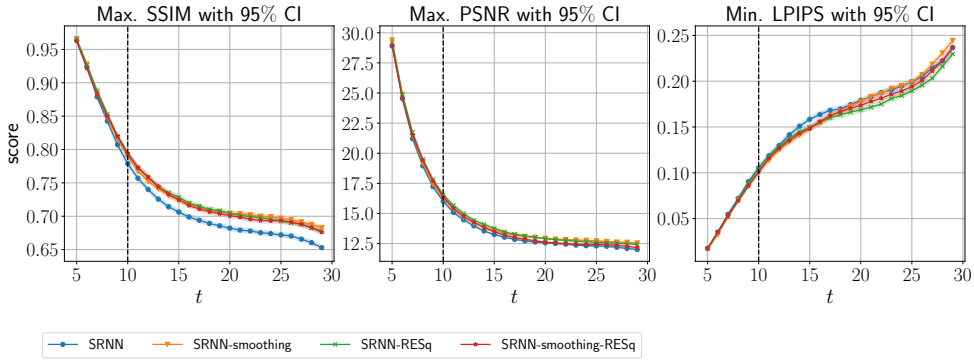


Figure 9.1: Max/Min similarity measures over different regularisation methods for the SRNN. For every time step the measures are sampled 100 times. The vertical line indicates number of frames the model is trained on.

It is seen that there is minor differences between the regularisation methods. Based on the SSIM in Figure 9.1 it might be beneficial to use smoothing or RES$_q$. Including both does not seem to improve the results significantly.

The latent overshooting, as described in Section 3.6, is investigated using the SM-MNIST data set. In Figure 9.2 the results of the latent overshooting experiments are shown. No significant difference is seen between the experiments. This is in accordance with Hafner et al. [29], latent overshooting do not improve the predictions of the model.

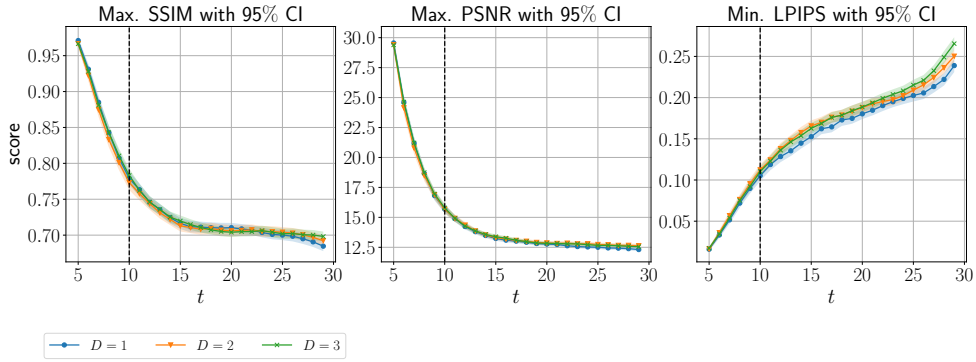


Figure 9.2: Max/Min similarity measures using latent overshooting $D \in \{1, 2, 3\}$ for the SRNN model. Each measure is sampled 100 times. $D = 1$ is equal to no latent overshooting.

9.2 RFN experiments

In this section experiments on the RFN are presented. With the SM-MNIST data set the effect of the skip connections between the different layers of the model are investigated. Four models with different skip connections from the feature extractor are investigated: skip connections to the conditioning network, skip connections to the CNF, skip connections to both, a model with no skip connections. The max/min similarity measures of a generated sequence with a warm-up on 5 frames. As seen in Table 9.1, one disadvantage of giving the model skip connections as a condition to the CNF is that the number of parameters of the flow increase with K. It increases the computational and memory demand of the model. The number of parameters increase in a more controlled manner when only using skip connections to the conditioning net.

Skip Connection	# Parameters	Validation BPP
No skip	$\approx 33 \times 10^6$	0.940 ± 0.024
Conditioning network	$\approx 35 \times 10^6$	0.946 ± 0.019
CNF	$\approx 58 \times 10^6$	0.991 ± 0.021
Conditioning network & CNF	$\approx 60 \times 10^6$	1.017 ± 0.030

Table 9.1: RFN with different skip connections trained on SM-MNIST, with $L = 5$ and $K = 10$. The bits-per-pixel is for the validation set with one standard deviation. Lower is better.

The results of the predictions against the ground truth values are seen in Figure 9.3, with a figure showing the max./min. similarity measures. A significant improvement in all measures can be seen with the included skip connections. The same large improvements cannot be seen in the validation loss when looking at Table 9.1. Here, only a minor difference is noticed between the different skip connections. A possible explanation to this is that the skip connections do not seem to carry over any significant improvement to the models' ability of reconstructing the digits, but only the models' ability to predict. Another factor to take into account is the large increase in the number of parameters introduced with skip connections. The large increase of complexity of the model, introduced with the extra parameters, might be the reason the model sees an improvement with both skip connections to the flow and conditioning network versus only the conditioning network. As such, even though an improvement is seen using skip connections in both the CNF and conditioning network, it is not significant enough to justify the large amount of extra parameters. This is justified by a minor difference in the measures when only using skip connections to the conditioning network.

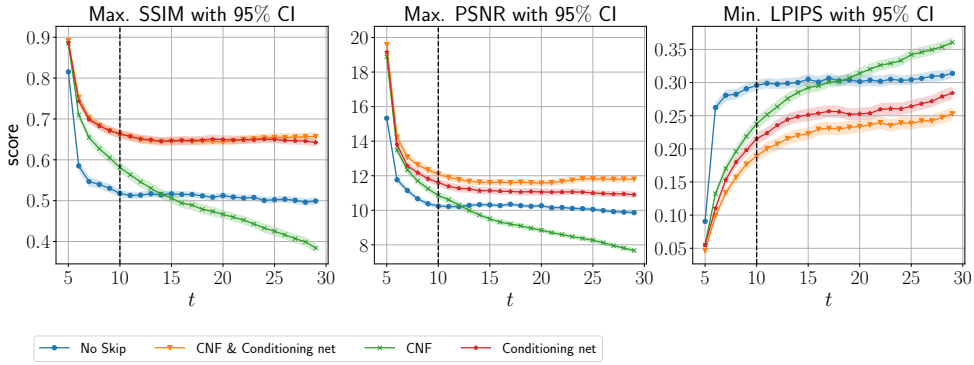


Figure 9.3: Max/Min similarity measures over the different skip connections for the RFN. Scores are sampled 100 times.

To investigate the depth, L, of the flow, three models are trained on the BAIR data set. From Table 9.2 the increase of parameters and the validation loss can be seen. A sharp increase in the number of parameters is noticed when increasing the depth, but from Figure 9.4 the difference in quality of the predicted frames is minor. The reason for the larger increase of parameters between $L = 4$, $L = 5$ versus $L = 3$, $L = 4$ is that the channels of skip connections also increase along with the depth of the flow. For the BAIR data set $L = 4$ seems to perform slightly better in the LPIPS measure, while having a lower number of parameters compared to $L = 5$.

Model	# Parameters	Validation BPP
$L = 3$	$\approx 22 \times 10^6$	2.654 ± 0.0211
$L = 4$	$\approx 36 \times 10^6$	2.636 ± 0.0212
$L = 5$	$\approx 61 \times 10^6$	2.665 ± 0.0216

Table 9.2: Different flow depths, L, for a RFN model trained on the BAIR data set. The bits-per-pixel is for the validation set with one standard deviation. Number of flow steps $K = 5$ are used. No skip connections to the flow, only to the conditioning net.

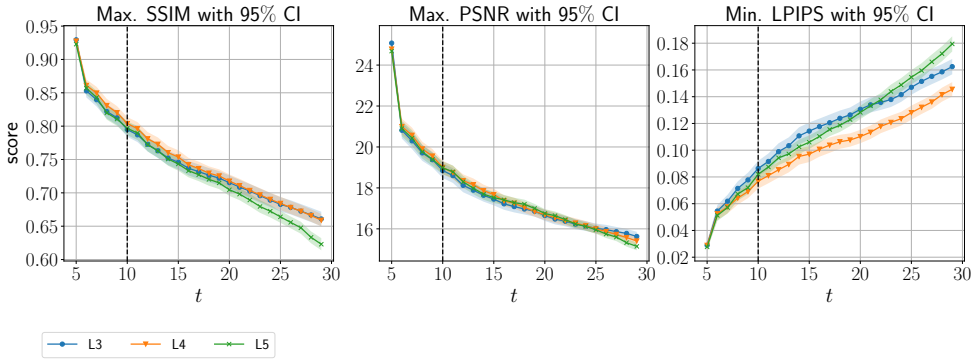


Figure 9.4: Max/Min similarity measures over the different types of skip connections for the RFN model. The scores are sampled 100 times.

The temperature does not seem to have a large impact on the predictions when looking at Figure 9.5, but a minor improvement is seen when using a temperature $T = [0.3, 0.7]$. Because a perceptual quality difference in the predictions is noticed when using a temperature factor, see Figure 10.1, a more thorough investigation to evaluate the temperature is conducted. Using a small validation set for the BAIR data set, the FVD is found for different temperatures. The results can be seen in Table 9.3. The FVD scores show a significant difference. The best FVD score is found using a temperature of $T = [0.5, 0.7]$. SSIM, PSNR and LPIPS do seem to emphasise the background and as the temperature mainly seems to affect the robot arm, this might explain no significant difference is observed in Figure 9.5. This also leads to a discussion about the commonly used evaluation metrics in Chapter 11. The effect of the temperature will be investigated in Chapter 10. A test using a temperature in the variational prior was also conducted. However, scaling the variance term of the recurrent variational prior does not benefit the predictions, see Appendix, Figure A.6.

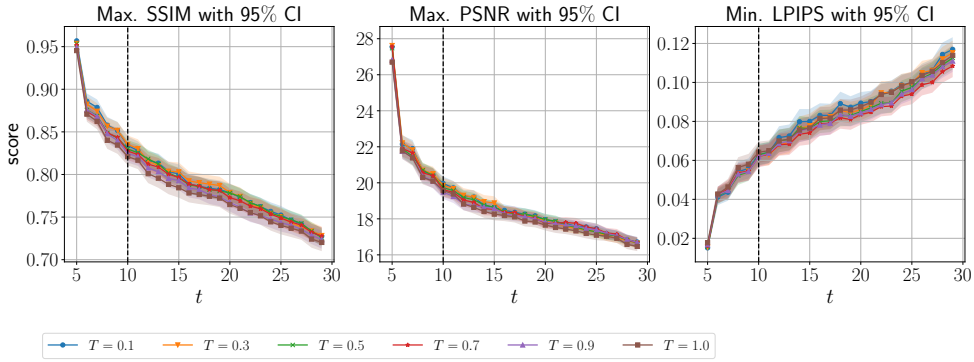


Figure 9.5: Max/Min similarity measures over the different temperatures on the vaildation BAIR data set for the RFN model, all values are sampled 100 times.

Temperature	FVD score	STD
0.1	217.3	±5.6
0.3	212.9	±4.2
0.5	204.5	±4.5
0.7	200.9	±3.7
0.9	212.6	±4.2
1	233.2	±4.8

Table 9.3: FVD score with a standard deviation on a small BAIR validation set. The model uses 5 frames in the conditioning to warm up and predicts 13 time steps into the future to evaluate against the ground truth. Each score is sampled 5 times.

9.3 Results - SM-MNIST

The results of a RFN trained on the SM-MNIST data are presented in this section. Additional results can be seen in Appendix, Section A.8.

In Figure 9.6 a series of generated predictions are shown. Predicted frames are highlighted with a green border of the image, while the red borders are the conditioned ground truth frames used as warm-up. It is seen that the RFN is only capable of generating visually pleasing digits for the one step predictions. After the first prediction the model struggles. The model is especially struggling to predict the overlapping behaviour of the digits. This behaviour is consistent with what was achieved in Kumar et al. [1], here strange behaviour is also observed when digits are close to overlapping. The problems with overlapping will be elaborated on in Chapter 11.

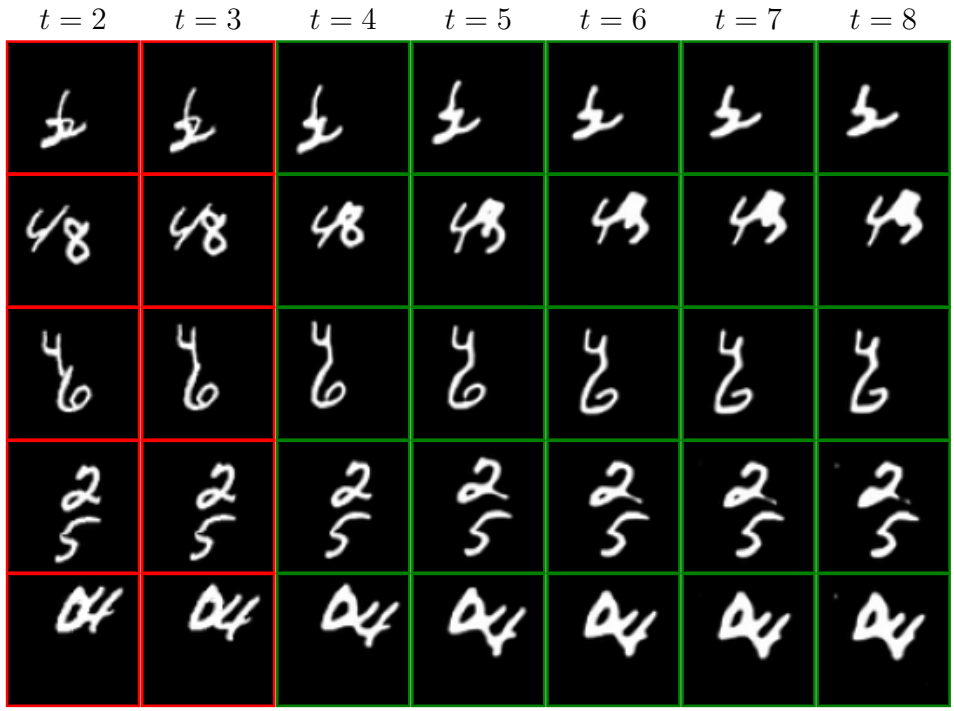


Figure 9.6: 5 predictions each conditioned on the previous frame. A warm-up period on 4 frames of different video sequences from the test set is used. Red border highlights the last 2 ground truth frames used in the warm-up. The green border highlights a prediction conditioned on the previous frame. Notice the poor behaviour when the digits overlap.

In Figure 9.7 the RFN is compared against the SRNN and VRNN model, which are using discretized mixture of logistics with the min./max. similarity measures. In the figure SRNN and VRNN are outperforming the RFN model. The poor result of the RFN is contributed to the fact that it struggles to model overlapping digits.

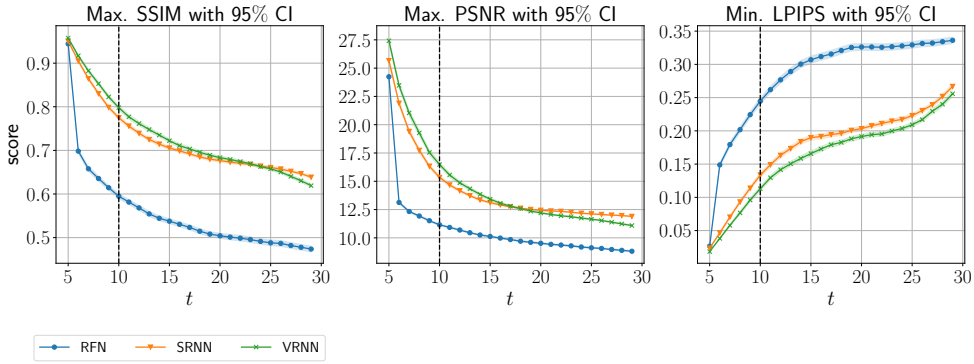


Figure 9.7: Min./Max. similarity measure between the SRNN, VRNN and the RFN on the SM-MNIST test data set. The vertical line represents the number of frames seen by the model during training.

In Table 9.4 the FVD is shown for the trained models, SRNN, VRNN, and RFN. They are all compared against different models. SRNN and VRNN both perform well and beat the RFN by a large margin on the SM-MNIST data set. SRNN seems to be slightly better than the VRNN. This is unsurprising given the RFN did struggle on the SM-MNIST data set.

# Frames seen: Training		
Conditioning	5	1
Total	15	10
# Frames seen: Evaluation		
Warm-up	5	5
Total	25	25
Model	**FVD**	
RFN - Ours	–	633.44 ± 4.20
SRNN - Ours	–	245.22 ± 3.27
VRNN - Ours	–	274.20 ± 4.17
VRNN-H [65]	57.17	–
SVG-LP[52]	90.81 [65]	–

Table 9.4: Different FVD scores for the SM-MNIST data set. For double cited results the first citation is the reference to the author of the model and the second is where the metric was found. "Frames seen: Training" explains how many frames the algorithm was directly conditioned on during training, for our models this is 1 with a total sequence length of 10. "Frames seen: Evaluation" describes the number of frames in the warm-up period before starting to predict up to the total amount of frames.

In Table 9.5 the BPP is shown for our models. VRNN and SRNN both get a lower BPP than the RFN. It highlights that SRNN and VRNN are better at estimating the underlying distribution and able to reconstruct the digit. No significant difference is seen between the VRNN and SRNN. This is expected, as the RFN did not do a good job modelling the SM-MNIST data set.

Model	BPP
RFN - Ours	$\leq 0.841 \pm 0.019$
VRNN - Ours	$\leq 0.546 \pm 0.005$
SRNN - Ours	$\leq 0.542 \pm 0.007$

Table 9.5: Different BPP scores for models trained on the SM-MNIST data set. The model is trained on 10 frames and conditioned on 1. Lower is better. The ELBO for the SRNN and VRNN are found by using importance weighted sampling with $K = 20$ and a discretized mixture of logistics in the output distribution. Each BPP is estimated 5 times.

9.4 Results - BAIR action-free robot pushing

Results of a RFN trained on the BAIR data set are presented in this section. For the results a temperature of $T = 0.6$ is used. In Figure 9.8 the predictions, again highlighted by a green border, are shown. The RFN captures realistic movement of the robot arm by maximising the log-likelihood of the current frame given the previous frame. It is seen that the robot arm follows a realistic behaviour based on the previous frames most of the time. Stochasticity is observed in the generated predictions, but the predictions do not seem to be very diverse. A plausible explanation to this is that the robot arm does not move very much during the 5 time steps, making it difficult to identify the diversity. The quality of the predictions are fairly consistent over time, but a small degradation in the quality of the robot arm is noticed. In the Appendix, Figure A.13 and Figure A.14 examples of predictions with worst and best SSIM score can be seen.

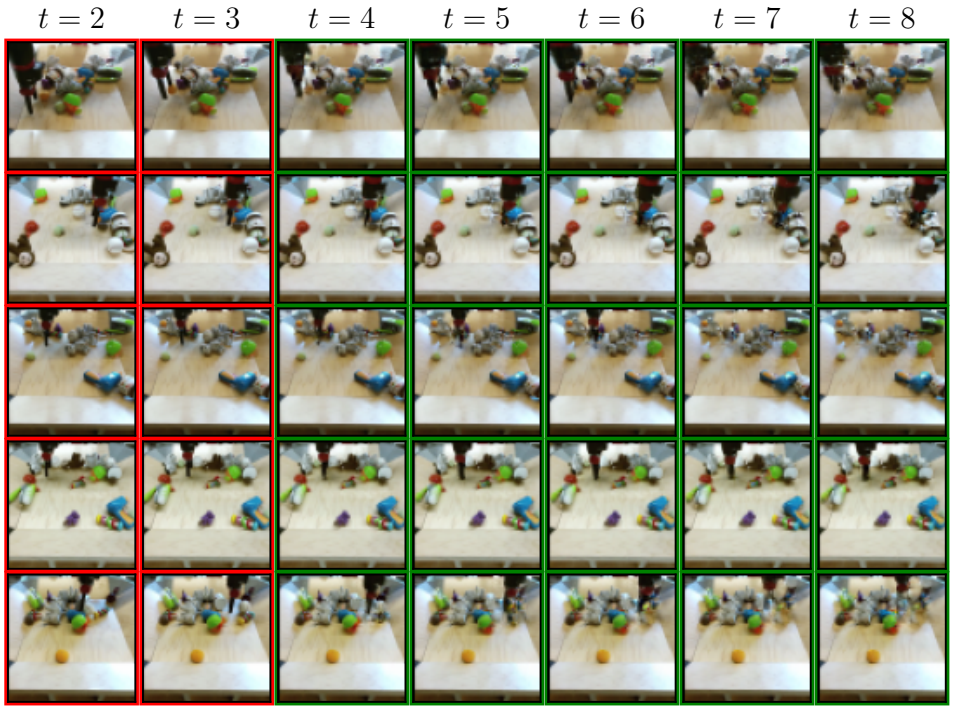


Figure 9.8: 5 predictions each conditioned on the previous frame. A warm-up period on 4 frames of different video sequences from the test set is used. Red border highlights the last 2 ground truth frames used in the warm-up. The green border highlights a prediction conditioned on the previous frame.

Figure 9.9 shows different stochastic outcomes of the model conditioned on the same initial sets of frames, showcasing the diversity of the predictions. It is seen that all sampled futures show probable outcomes, and that all of the futures are different. The RFN seems to capture the stochastic temporal dynamics of the robot arm well. The stochastic modelling of the RFN will be further elaborated on in Chapter 10.

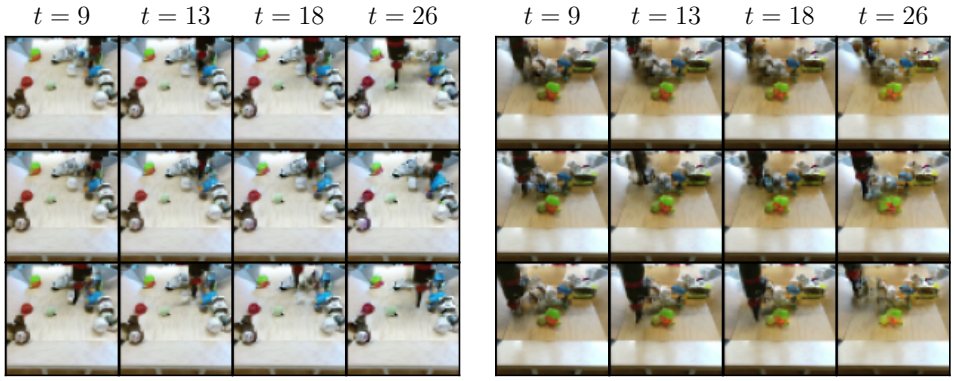


Figure 9.9: Displaying three different predictions for two sets of the same initial conditioning frames (left and right). The figures showcase the diversity in outcomes.

To see if the predictions are robust and consistent in quality over time, long range generations are shown in Figure 9.10. It is seen that even for $t = 70$ the robot arm is still generated, but there is a significant degradation in the quality of the robot arm and background over time. Looking closely it is possible to see artifacts and noise appearing in the background. When doing long range predictions the model needs to be stable and the previous predictions have to be excellent. Otherwise, noise and minuscule artifacts of the previous frames will accumulate in the predictions, as seen in Figure 9.10. It is noteworthy to mention that the robot arm sometimes disappears (see 3 row, $t = 40$). This is because the model struggles to capture the minor details, such as when the robot arm is very close to the edge and only the tip of the arm is shown.

In Figure 9.11 min./max. similarity of SRNN, VRNN and RFN models are compared. All models directly estimate the conditional output probability. The RFN model is performing significantly better in all three tests. It is found that in general the RFN favours RGB data, an explanation of this is given in Chapter 11.

In Table 9.6 the FVD score of the RFN is compared against competing models. VideoFlow and SAVM are superior based on FVD. The RFN does seem to be able to compete with other good performing models, such as hierarchical VRNN, SVG-LP

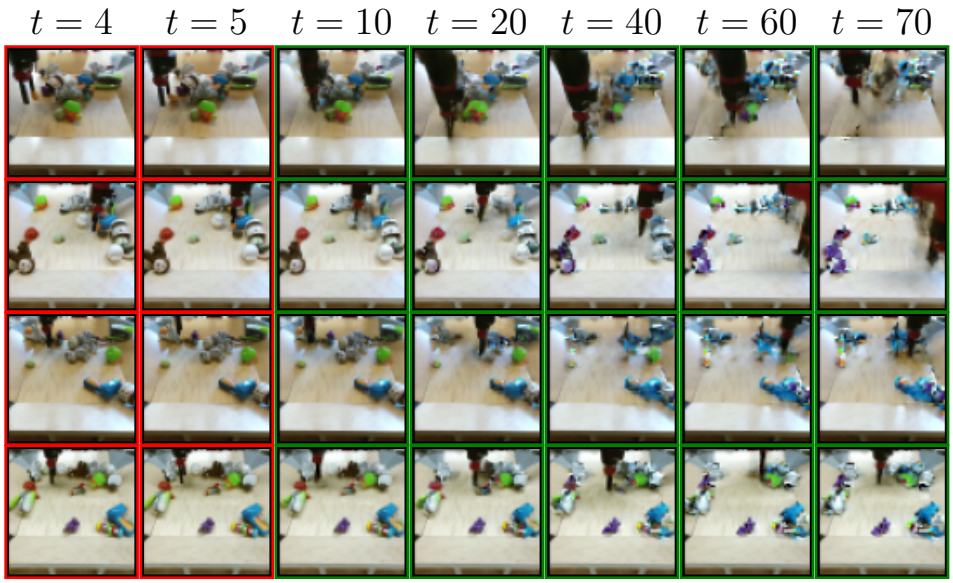


Figure 9.10: Long range predictions of different sequences conditioned on the previous frame, warmed up on 5 frames. Red highlights the last 2 ground truth frames used in the warm-up. Green highlights the predictions.

and SAVP. The SRNN and VRNN are both performing better than the RFN. However, it is noticed that the SRNN and VRNN have trouble estimating the variability of the data in the output distribution: the robot arm is seen, but the predictions are pixelated highlighting the uncertainty of the model. Again, it is noteworthy to mention that the RFN is trained on less total frames and only conditioned on a single frame during training, this should be taken into account when comparing the models.

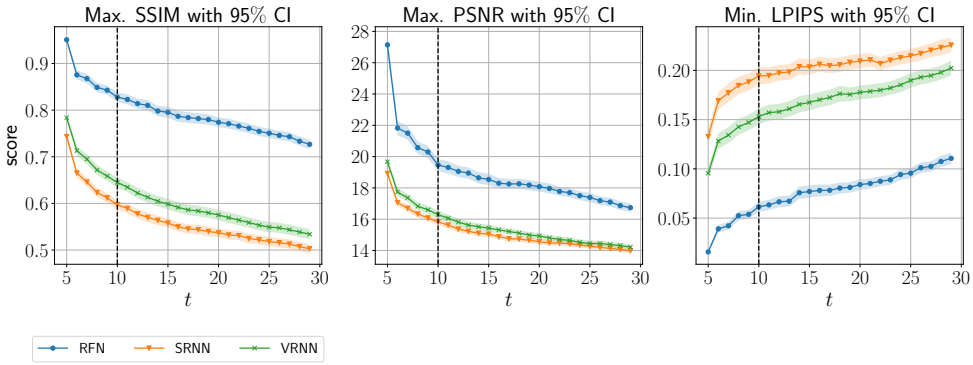


Figure 9.11: Min./Max. similarity of the SRNN, VRNN and the RFN model, on the BAIR test data. The vertical line representing the number of frames seen by the model during training.

		# Frames seen: Training		
Conditioning	3	2	2	1
Total	13	12	16	16
		# Frames seen: Evaluation		
Warm-up	3	2	2	1
Total	13	30	16	16
Model		**FVD**		
VideoFlow[1]	95 ± 4	–	–	–
VRNN-H [65]	–	143.4	–	–
SVG-LP[52]	–	256[65]	–	–
SAVP-VAE[66]	–	143[65]	116 [1]	–
SV2P[57]	–	–	263 [1]	–
SAVM [67]	–	–	–	94 ± 2
RFN - Ours	153 ± 4	283 ± 3	171 ± 2	181 ± 3
SRNN - Ours	721 ± 7	–	–	–
VRNN - Ours	738 ± 8	–	–	–

Table 9.6: Different FVD scores for the BAIR data set. The temperature of the RFN was set to $T = 0.6$. All of our models are trained on 10 total frames with only the previous frame as condition. The models predict up until time step 13, 15, 16 and 30 with a warm-up of 3, 2, 2, and 1, respectively.

In Table 9.7 the BPP of the RFN is compared against SRNN, VRNN, and other generative models. It is seen that the BPP of RFN is lower than SRNN, VRNN, SAVP, and SV2P, but it does not beat VideoFlow or SAVM. VideoFlow and SAVM are better at estimating the underlying distribution of the data.

Model	**BPP**
RFN - Ours	$\leq 2.16 \pm 0.02$
SRNN - Ours	$\leq 3.701 \pm 0.001$
VRNN - Ours	$\leq 3.576 \pm 0.002$
SAVM [67]	$= 1.35$
VideoFlow [1]	$= 1.87$
SAVP-VAE [66]	≤ 6.73 [1]
SV2P [68]	≤ 6.78 [1]

Table 9.7: BPP scores for the BAIR data set when evaluated on 10 target frames in a sequence. For VideoFlow and SAVM the BPP is exact. BPP for SAVP-VAE, SV2P and VideoFlow are evaluated over 10 target frames while conditioning on 3 frames. SAVM was conditioned on a single frame with 15 target frames. The ELBO of the SRNN and VRNN are found by using importance weighted sampling with $K = 20$. Each BPP is estimated 5 times.

9.5 Results - KTH

To give a better estimate of the predictive performance of the model the best and worst samples are shown in Figure 9.12 and 9.13, respectively. The model approximates the underlying data distribution and captures the movement of the subjects fairly well. However, it is not perfect and predictions of the subjects are seen to slowly wash out with finer details being lost, such as hands. For the worst predictions it is seen that the model struggles estimating the subjects that are boxing and lack defining features. The arm merging with the body resembles the same problem as seen in Figure 9.6 where two digits overlap for a short period. It is a problem that is particularly difficult for the RFN to handle.

The diversity of the predictions can be seen in Figure 9.14, where a set of predictions are seen with the same initial conditions. Due to the lack of quality in the images, such as missing limbs of the subject, it is difficult to see the impact of the stochastic dynamics. However, it seems that slightly different outcomes for the same initial frames are achieved. Notice how the model is keeping an almost constant velocity of the moving subject with only a minor deviation, which might be due to the accumulation of artifacts.

For the KTH test set the max./min. similarity score is shown in Figure 9.15, where all models directly measure the conditional output probability. It is seen that the RFN is scoring the best on the SSIM and LPIPS metric, while for PSNR, the SRNN and VRNN are initially scoring higher for the first predictions in the sequence. Several reasons could explain this, one of them might be that the slight pixelation of the SRNN and VRNN predictions are less penalised by the PNSR measure, which favours matching temporal dynamics higher. As such, the VRNN and SRNN might have performed modelling the temporal dynamics better than the RFN, hence the predictions are seen to have a higher score.

In Table 9.8 the FVD score of the RFN is compared to other models. Here, it is seen that the RFN performs better than VRNN, SV2P, and SRNN, based on the FVD score. However, VRNN-H and SVG with a learned prior still achieves a significant lower score than the RFN. With the gap between the models and the RFN score likely being smaller than what is shown. The RFN is only trained on 10 total frames versus the other models being trained on 20 frames, causing the metric score to not be directly comparable.

In Table 9.9 the BPP of the RFN is compared to SRNN and VRNN. The SRNN and VRNN have a higher BPP than the RFN. The RFN seems to perform significantly better than the SRNN and VRNN.

# Frames seen: Training		
Conditioning	3	1
Total	20	10
# Frames seen: Evaluation		
Warm-up	10	10
Total	40	40
Model	**FVD**	
RFN - Ours	–	539 ± 6
SRNN - Ours	–	987 ± 6
VRNN - Ours	–	968 ± 8
SV2P [57]	636 ± 1 [60]	–
SAVP-VAE [66]	374 ± 3 [60]	–
SVG [52]	377 ± 6 [60]	–
SLRVP [60]	222 ± 3	–

Table 9.8: Table of different FVD scores for the KTH data set. For double cited results the first citation is the reference to the author of the model, and the second is where the metric is found.

Model	BPP
RFN - Ours	$\leq 2.663 \pm 0.045$
SRNN - Ours	$\leq 3.982 \pm 0.062$
VRNN - Ours	$\leq 4.110 \pm 0.109$

Table 9.9: BPP scores for the KTH data set. Models are trained on 10 frames and conditioned on 1. The ELBO for the SRNN and VRNN are found by using importance weighted sampling with $K = 20$. Each BPP is estimated 5 times.

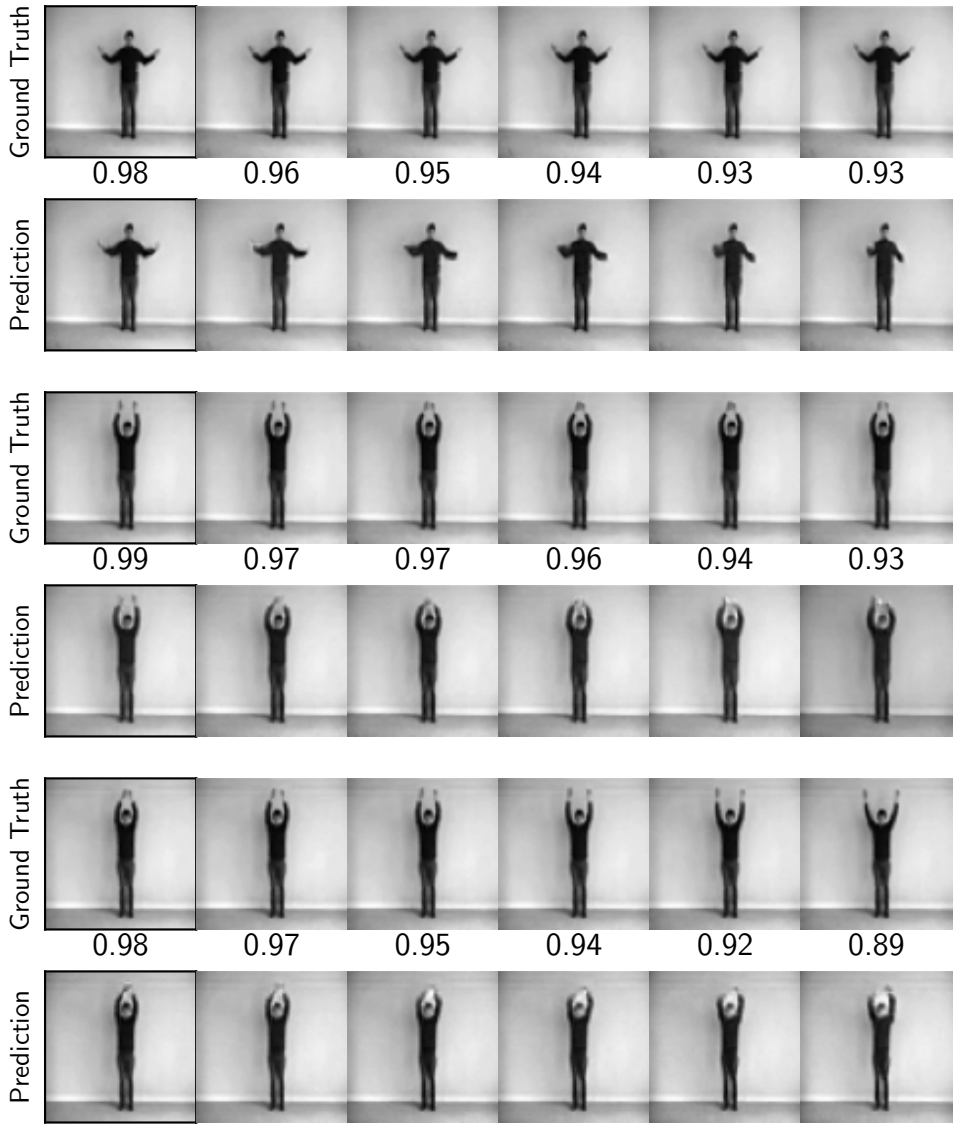


Figure 9.12: Predictions of the best samples based on SSIM for different video sequences. The figures alternate between the ground truth and its corresponding prediction.

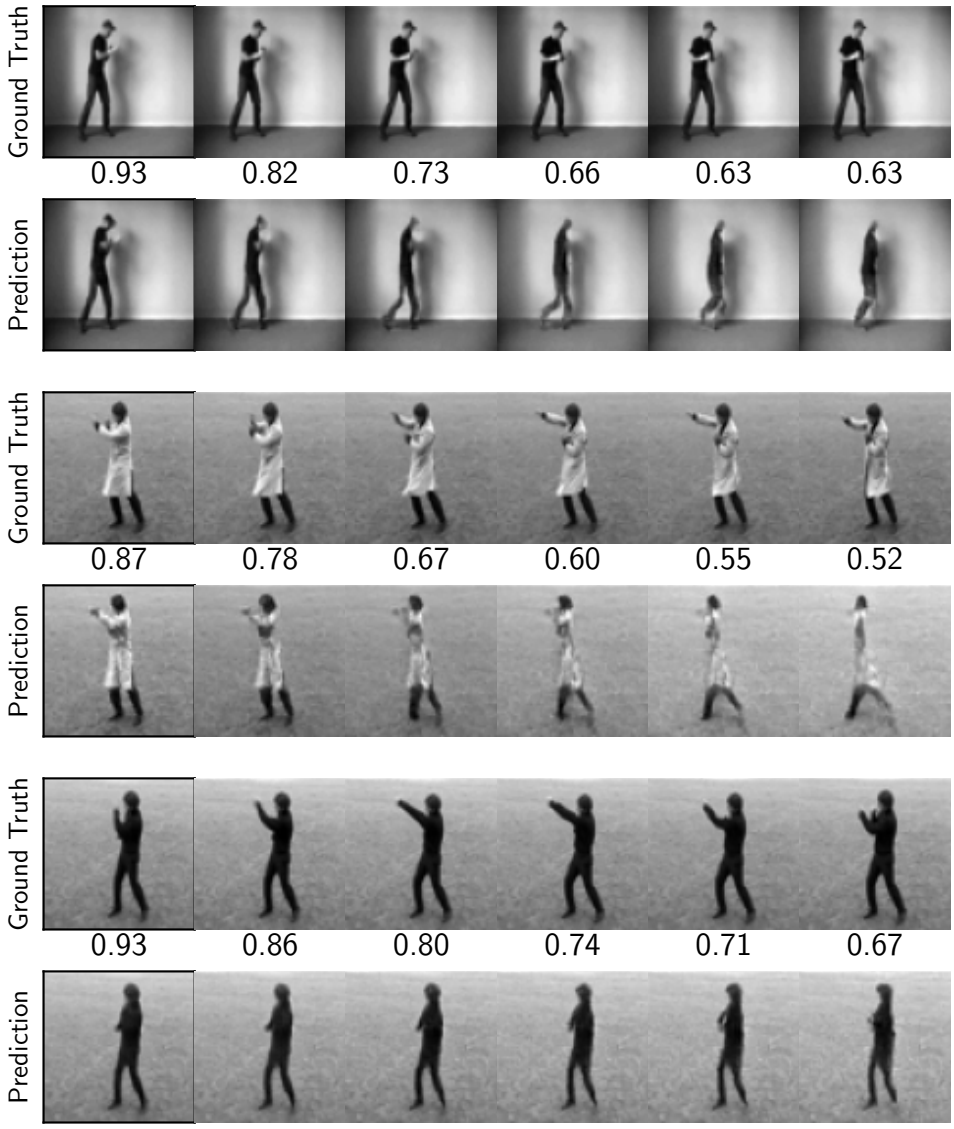


Figure 9.13: Predictions of the worst samples based on SSIM for different video sequences. The value shown is the SSIM score between the ground truth and its corresponding prediction.

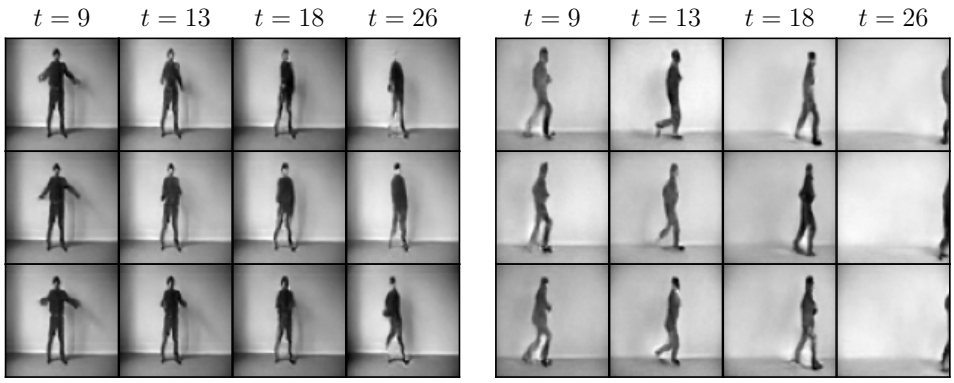


Figure 9.14: Displaying three different predictions for two sets of conditioning frames (left and right). The figures showcase the diversity in outcomes. A temperature $T = 0.6$ is used.

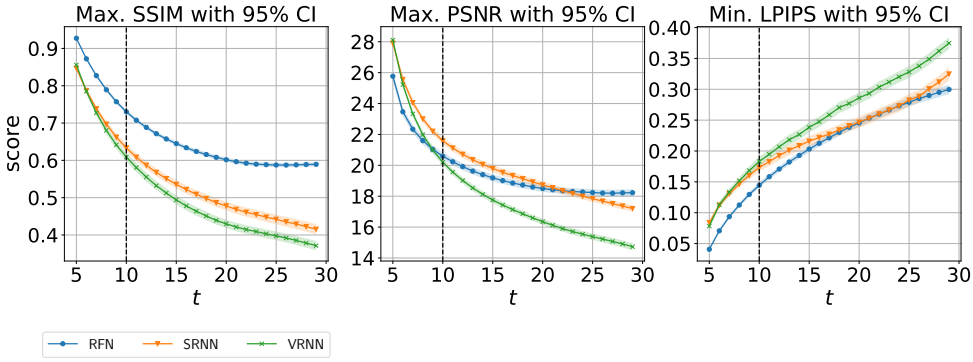


Figure 9.15: Min./Max. similarity measures of the SRNN, VRNN and the RFN model on the KTH test data set. The vertical line represents the number of frames seen by the model during training.

CHAPTER 10
Analysis of the RFN

In this chapter an analysis of the RFN trained on the different data sets is conducted. It will present various figures containing different analytic results. The study is conducted to see if some of the assumed properties of the RFN are fulfilled. The assumed properties being examined are; a smooth latent space, a separation of the variational states, and parameters behaving as expected. Furthermore, the analysis is also conducted to showcase what is happening within the model, such as how the modelling of spatio-temporal dynamics are explained by the model, and the behaviour of the variational prior and posterior. The analysis might also assist in further explaining some of the issues of the model.

10.1 Temperature

While VideoFlow only had a normalising flow to model the stochastic dynamics of the data. The RFN has both a normalising flow and a variational temporal prior. In Kumar et al. [1] they argue that there is a trade-off between the temporal stochastic dynamics and noise level of the background based on the temperature factor in VideoFlow. We investigate if this is also the case for the RFN.

In Figure 10.1 different predictions at several time steps for different temperatures are seen. The standard deviation of the variational prior has been fixed to a very low number making $\boldsymbol{\sigma}_t^{(p)} \approx 0$. The standard deviation of the base distribution $\boldsymbol{\sigma}_{basedist}$ and the conditional split priors in the flow has then been factored by T. Looking at Figure 10.1 it is seen that the robot arm is showing minimal movement for all the shown time steps up to $t = 40$. However, the noise level of the robot arm and for the objects in the background seems to vary. For higher temperature of the base distribution the noise and number of artifacts seem to rise, which is in accordance with the findings in Kumar et al. [1]. If the temperature is too high, artifacts and noise start to appear on the objects in the background and the robot arm.

To see if the hypothesis is correct for the variational prior, we do the same thing as previously, but this time we make the standard deviation of the conditional split prior $\boldsymbol{\sigma}_{basedist} \approx 0$. The standard deviation $\boldsymbol{\sigma}_{prior}$ is then multiplied by the factor T. Looking at Figure 10.2, it is seen that for a higher T the robot arm is fairly erratic,

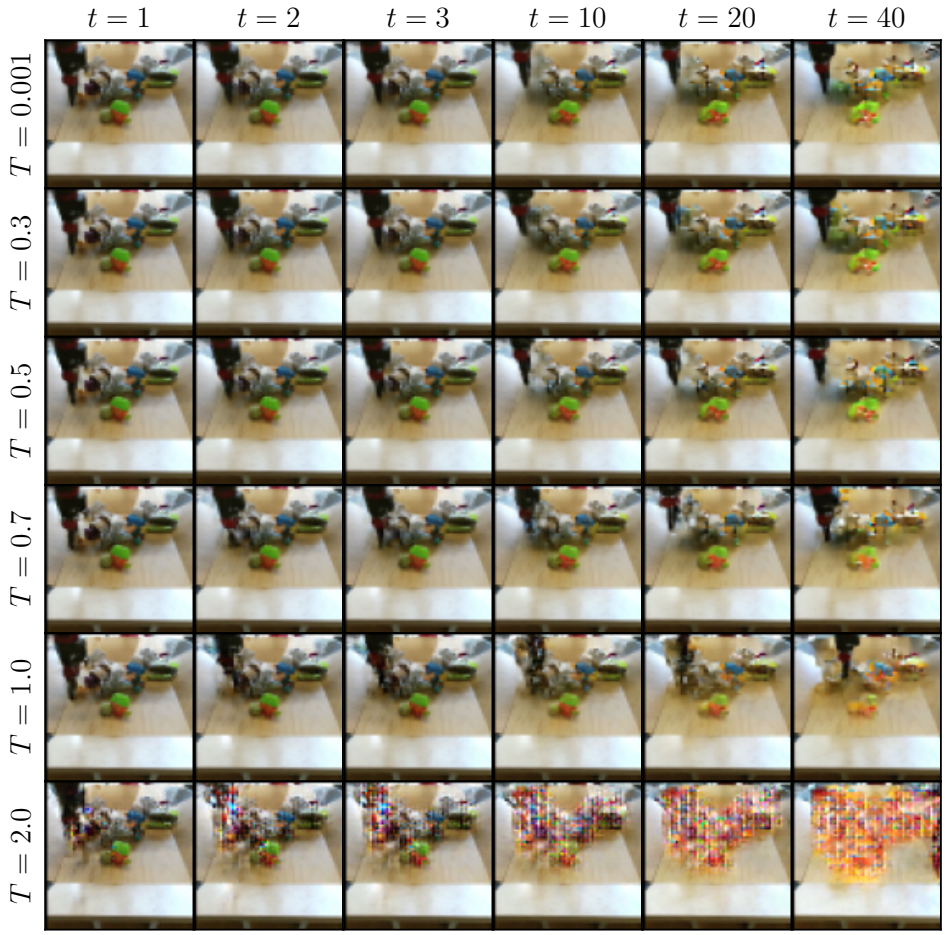


Figure 10.1: The effect of the temperature, T in the base distribution and the conditional split priors of the normalising flow. Predictions are done on the same initial sequence. The time t is seen above the figures and to the left the specified temperature T is seen. The standard deviation of the variational prior is fixed to a very low value.

and even quite blurry at times. The same artifacts seen with a high T in the base distribution of the flow is not observed. Therefore, with a high temperature T in the variational prior, the temporal stochastic dynamics and diversity are increased. Meanwhile, for a high T in the base distribution of the flow increases spatial noise and artifacts.

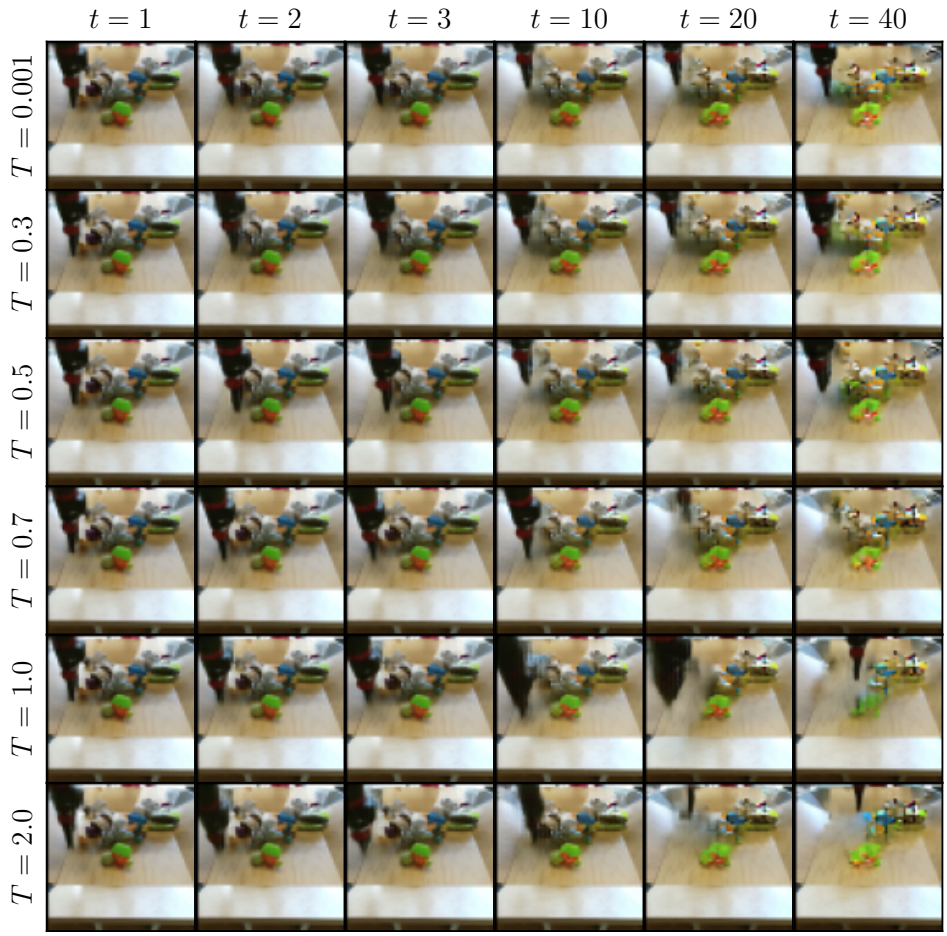


Figure 10.2: The effect of the temperature, T in the variational prior of the normalising flow. Predictions are conducted on the same initial sequence. Time t indicated top of figure and specified temperature T on the left. The base distribution standard deviation of the normalising flow is fixed to a very low value.

As we want high diversity σ_{prior} is typically not scaled in the literature [1], this is also confirmed in our findings, shown in Appendix A.6. We did see a performance gain for a lower $\boldsymbol{\sigma}_{basedist}$ and standard deviation in the conditional split priors, as was also found in Table 9.3. Based on the findings above, we reckon that the stochastic temporal dynamics are mainly captured in the bottleneck of the RFN, i.e. by the

variational prior and posterior. Further, we argue that the conditional normalising flow of the RFN, models the spatial variability of the data, i.e. the spatial stochastic dynamics.

10.2 Interpolation

An interpolation over the latent space defined by $(\mathbf{h}_t, \mathbf{z}_t)$ between two sequences are depicted in Figure 10.3. The interpolations are done for a RFN trained on a single digit. The interpolation is done by reconstructing two sequences from the test set with the same initial frame. From the reconstruction $\mathbf{z}_t, \mathbf{h}_t$ every time step is stored. The shown interpolations in the figures are then the reconstructions of the interpolated latent features between the two initial reconstructed sequences.

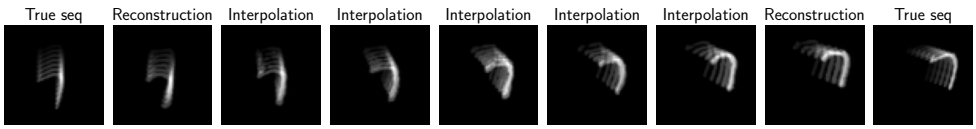


Figure 10.3: RFN model trained on one digit from the SM-MNIST data set. An interpolation between two sequences conditioned on the same initial frame is presented. The fading of the digits represents the sequential frames, where the most faded digit is the first generated frame in the sequence.

It is observed the interpolation between the two sequences show expected behaviour, as the velocity vector seems to gradually shift over the interpolations between the two reconstructed sequences. As the interpolations seem consistent it is assumed the latent space is well behaved and smooth for a single MNIST digit [6]. This assumption cannot necessarily be generalised to other data sets, but we see no reason why this is not also the case for BAIR and KTH given the promising results.

10.3 Difference between prior and posterior

To investigate where the variational prior of the models have a difficult time following the variational posterior, the variational gap for each prediction in Figure 10.4 is presented. Here, one step predictions of the prior and the reconstruction are shown together with the ground truth sequence. It is seen that the reconstructions closely resemble the ground truth, and only a slight deviation is noticed. A large deviation is seen when two digits are overlapping. As expected, the prior samples have less visual quality than the posterior encoded samples. It is seen that the KL divergence is highest whenever a number is at the border of the frame or is the initial generated image. This is expected, as stochastic behaviour of the data set is only present at

the border of the image. This is where the prior will have to mimic the variational posterior and not only be able to rely on the encoded features of $\mathbf{h}_t$.

The last row shows the negative log-likelihood of the flow distribution in BPP over the prior and the posterior given the ground truth to t. The log-likelihood of the posterior is $\mathbb{E}_{q_\phi(\mathbf{z}_{\leq t}|\mathbf{x}_{\leq t})}\Big[\log p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right)\Big]$ and the log-likelihood for the prior is $\mathbb{E}_{p_\phi(\mathbf{z}_{\leq t}|\mathbf{x}_{<t})}\Big[\log p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right)\Big]$.

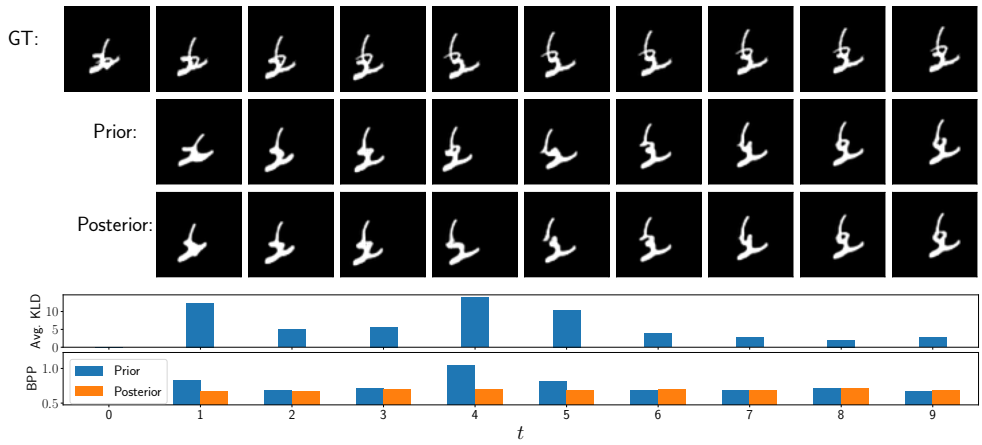


Figure 10.4: A series of figures showing the difference between the predictions generated with the variational posterior and the variational prior. First row: reference ground truth. Second row: one step predictions of the prior. Third row: reconstruction of the posterior. Fourth row: KL divergence of the time-step. The figure at the bottom shows two negative log-likelihoods of the ground truth, for the flow distribution conditioned on the variational prior or the variational posterior distribution.

It is seen that a spike in the KL divergence is correlated with a difference in BPP between the posterior and prior. This difference is expected as the KL divergence measures the difference between the variational prior and the approximate posterior in the spatio-temporal latent space. Whereas, the negative log-likelihood is an exact measure for how the model performs, as such it is expected there is some correlation between KL divergence and a difference in BPP for the posterior and prior distribution. Additional figures of this kind for the different data sets can be seen in the Appendix A.7.

10.4 Spatio-temporal distribution

To estimate how good the RFN model is able to explain the spatio-temporal distribution the average negative log-likelihoods over the test set are found for a flow distribution of a trained RFN conditioned on 4 frames, see Figure 10.5.

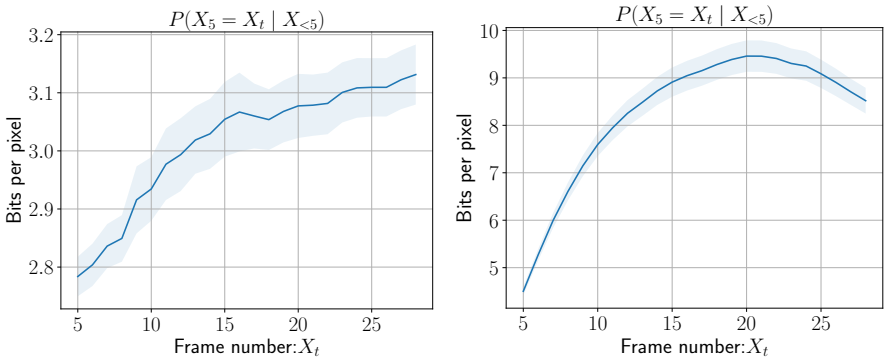


Figure 10.5: The BPP of the ground truth images given a flow distribution conditioned on 4 frames averaged over the whole test set, for the BAIR data set (left), and the KTH data set (right). A lower bits-per-pixels imply a more probable frame.

The NLL is found for the ground truth $\mathbf{x}_t$, over the conditional flow distribution $\mathbb{E}_{p_\phi(\mathbf{z}_{\leq 5}|\mathbf{x}_{<5})}\Big[-\log p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_{\leq 5}, \mathbf{h}_{\leq 5}\right)\Big]$ for $t = [5, 29]$, when not updating the conditional flow distribution over the time steps. The expected behaviour for the NLL of the future frames, is that the NLL will monotonically increase with t, as then the model have a "sense" of the spatio-temporal distribution. This monotonic behaviour is observed for the BAIR data set in Figure 10.5 (left). Concluding, the model seems to have an understanding of the spatio-temporal distribution.

In Figure 10.5 (right) for higher t the NLL decreases. This is expected to be a data set dependent consequence as the KTH data set is somewhat cyclic. Some sequences of the KTH data set will end and start with the same empty background, giving notice to the observed effect.

10.5 Bits-per-pixel and image quality

To give a sense of how the loss affects the quality of images, the corresponding BPP of a sequence is shown in Figure 10.6.

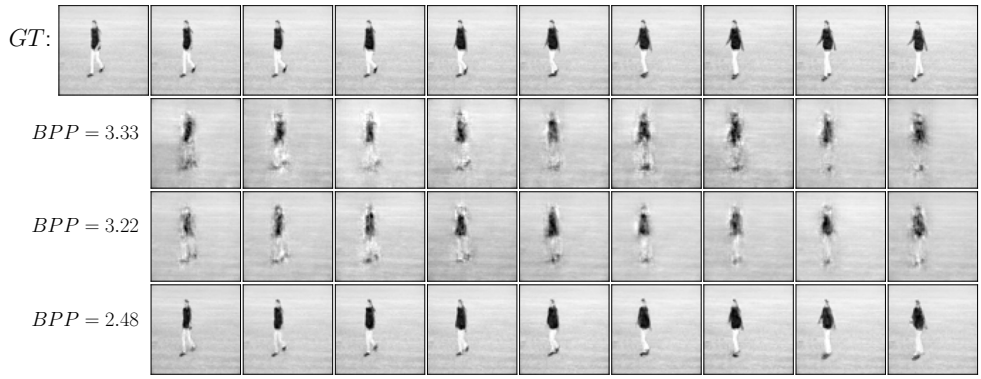


Figure 10.6: The ground truth (GT) and one step predictions of the prior and the associated bits per pixel (BPP) average over the sequence.

A better BPP correlates with an improved quality in the frames. Even a small improvement of 0.1 BPP seems to make a slight improvement. A significant improvement is seen for the lowest BPP.

10.6 Parameter behaviour

To analyse how the RFN models the stochastic dynamics of the SM-MNIST data, an experiment for a synchronised moving SM-MNIST data set is conducted. In the synchronised moving SM-MNIST data set the digits differ, but the movement of the digits over the sequences are set to the same seed. For these sequences a select set of parameters are shown to illustrate their temporal variations. In Figure 10.7 these parameters and their average over time is seen. The parameters shown are the mean and standard deviation of the variational prior, variational posterior, and the base distribution of the flow. The vertical lines highlight if a digit collides with a boundary, with red and blue highlighting each digit.

Temporal stochasticity in the data set only occurs when the digits bounce off the border of the frame, causing the parameters to spike. The spike behaviour highlights that the model captures the stochastic bounce of a digit. It is also noteworthy, that the standard deviation continually remains large between some of the spikes, which is when the digits overlap. So, although the overlapping of digits is not a stochastic

process, the model treats it as such. This wrongful treatment of stochasticity might explain the erratic behaviour when digits overlap, as the RFN models the overlapping by a stochastic process and not through the deterministic state. To further showcase the effect seen in the parameters when digits overlap, the same figure is shown, but only for a single digit. It is shown that the RFN is able to model the movement of a single digit well in Figure 10.7. For a single digit sequence no overlap between digits is possible, as such the standard deviation for the different parameters only increases when the digit hits the border, as seen in Figure 10.8.

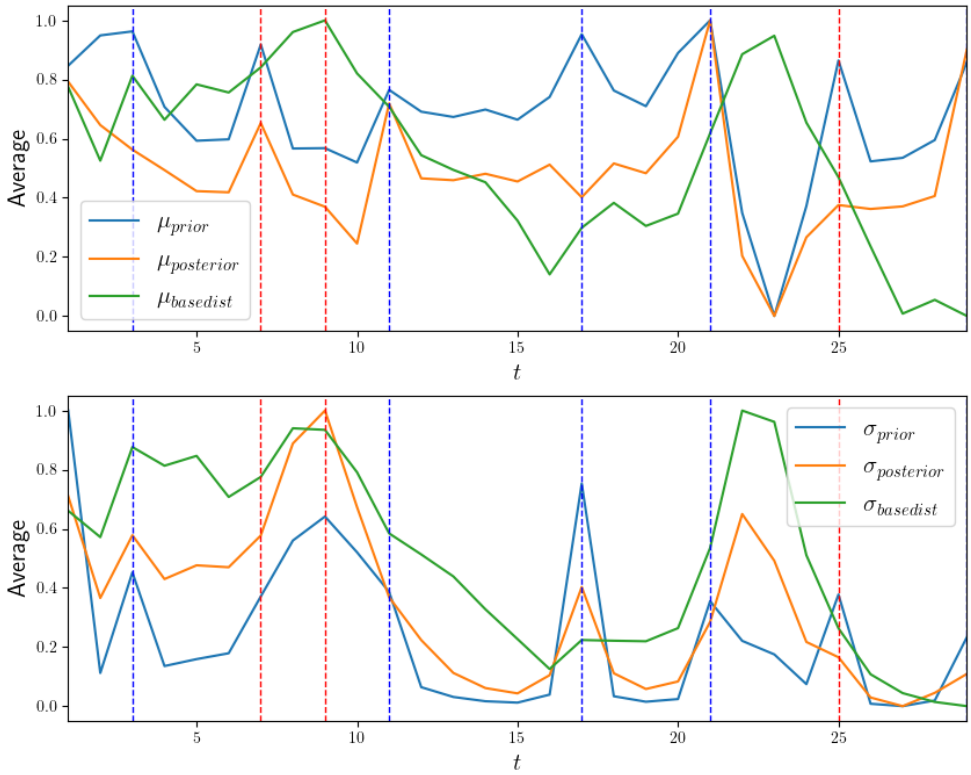


Figure 10.7: Average of normalised parameter values of the variational prior, variational posterior and base distribution of the flow. The values are found by fixing the movement of the digits throughout the sequences, only the digits will change among the different sequences. 400 different sequences of one-step predictions are shown and averaged over each time step until $t = 30$. The vertical lines highlights when a digit hits a boundary. Red highlights the first digit, blue highlights the second digit. In Appendix A.10 a single sequence of the used one step predictions for this figure can be seen.

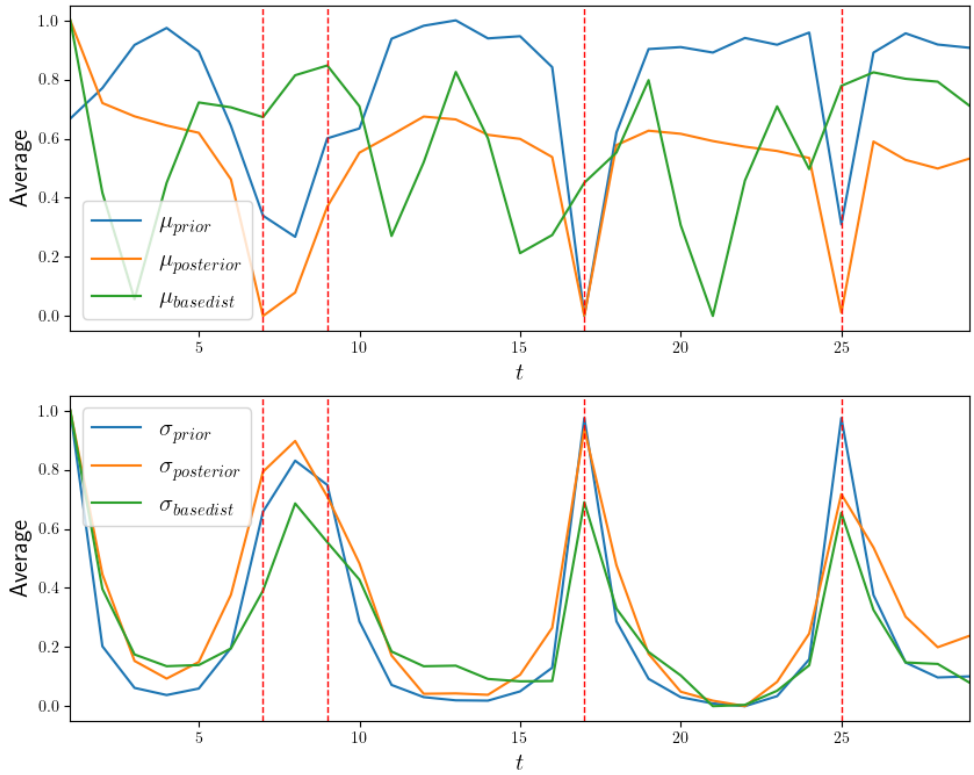


Figure 10.8: Replica of Figure 10.7, but now only a single digit is used.

CHAPTER 11

Discussion

In this section the obtained results will be discussed. A focus will be placed on; results of the regularisation methods, the different aspects of building a RFN model, advantages/disadvantages of the RFN, pitfalls, and the difficulties of the model.

Regularisation methods:
With the SRNN, different regularisation methods were tested. This testing was done to validate if the regularisation methods had an effect on the sequential image data, and whether it should be used in the RFN model. The different regularisation methods were: smoothing, latent overshooting and RES_q. Looking at the SSIM score only a minor improvement was seen on the ability of the SRNN to predict using smoothing or RES_q.

For the case of smoothing, we argue that the variational posterior of the model does not benefit much from having access to information from future frames. This might be because the posterior will be relative stationary in a sequential image data environment, as the tested data does not seem to have any long term correlations.

Conditioning the prior directly on the variational posterior for the previous time step, as done in RES_q and latent overshooting, did only seem to have a minor effect on the ability to predict the model, with latent overshooting not improving the performance. This might be a consequence of the variational prior only being trained on one-step predictions from the posterior, so that for a chaining of priors $p(\mathbf{z}_t|\mathbf{z}_{t-1})$, the prior might not be able to correct the inconsistencies from not being conditioned on the posterior.

The purpose of latent overshooting were to alleviate the issue of letting the variational prior be directly conditioned on the variational posterior from the last time step. For our experiments latent overshooting did not have any significant effect. This was in accordance with Hafner et al. [29]. No effect of overshooting can be contributed to the fact that the SRNN model separates the deterministic $\mathbf{h}_t$ and stochastic $\mathbf{z}_t$ states. This means that the variational prior and posterior are always conditioned on themselves, leaving the latent overshooting redundant.

Skip connections:
To get meaningful results it was crucial to use skip connections, as they let the inference model focus on the temporal dynamics and improved the prediction [60]. With the skip connections a trade-off must be made between complexity and performance, as was seen in Table 9.1. Using skip connections for both the CNF and the conditioning network achieved the best results, but the large increase in complexity could have biased the results in Table 9.1. From the huge additional memory consumption, computational demand, and not a very large increase in performance, an increase in parameters for both skip connections in the CNF and conditioning network did not seem to be worth it.

Aleatoric and Epistemic Uncertainties:
Modelling temporal dependencies can be a difficult and challenging task. In this thesis different generative models were explored, two of them being the SRNN and VRNN model. The main difference between the models are the SRNN is modelling the stochastic and deterministic part of the model separately, whereas the VRNN models the stochastic spatial dynamics directly determined by the deterministic parameter $\mathbf{h}_t$.

In Fraccaro et al. [3] an advantage of separating the stochastic and deterministic temporal states was discovered. The RFN model seeks to do the same separation of states, by letting the partition of the SRNN represent the stochastic temporal dynamics and deterministic dependencies of the data through $\mathbf{h}$ and $\mathbf{z}$. This separation of states was also seen in Section 10.1 when decreasing the variance of the variational prior distribution. The CNF partition of the model then represents the spatial variability. This is also seen in Figure 10.1, where reducing the variance of the base distribution and the conditional split priors in the flow, gradually smooths the image, making them less spatial diverse.

This could be seen as the RFN disentangles the spatial and temporal stochastic, with the SRNN taking part of the aleatoric uncertainty, e.g. the unknowns that differ over time, such as digits colliding with a border, or the robot arm moving to a new coordinate. While the CNF will model the epistemic uncertainty, which is caused by limited knowledge and data, e.g. the small unknown details of each subject in the KTH data set, observational noise etc.

Overlapping problems and lack of fidelity:
The RFN seems to have trouble modelling overlapping multi-modal distributions as seen when two numbers overlap in the SM-MNIST data set or when a subject is boxing in the KTH data set. As observed in other studies [37, 69] the Glow model have trouble modelling discontinues regions of low probability, making the model unable to clearly separate modes in the latent manifold. This poor modelling of low probability regions gives the Glow an opportunity to show tail like behaviour [69], which is also seen in simpler cases [5, 6]. In Tygesen [37] it is further discussed that affine coupling

layers are unable to create disjoint distributions. We think this is also the case here, i.e. a clear separation of modes in the latent manifold is required to properly model the overlap of digits and the boxing problem, when using normalising flows. We also contribute the Glow and its problems with modelling low probability edge cases to the lack of fidelity seen with missing arms of subjects. It is probable that fidelity is lost in the many convolutional layers used in the network structure, as the SRNN and VRNN also struggle with modelling the fidelity of subjects in the KTH data set.

The BAIR data set contains RGB images with a diverse colour palette and thus a more continuous range of pixel values compared to SM-MNIST. Furthermore, a single moving object is only present; the robot arm. In this case the RFN did show superior performance. As the single moving robotics arm does not require any disjoint distribution to model, and having a smoother and more continuous manifold in RGB space seem to be the ideal scenario for the RFN.

Temperature:
In Kumar et al. [1] it is argued that the temperature in VideoFlow introduces a trade-off between stochastic temporal dynamics and a reduction of background noise. The same trade-off is not observed in the RFN. We argue that this is because the stochastic temporal dynamics are mainly modelled by the variational distribution, which do not use a temperature factor. We did notice a reduction in the quality of the images and an increase in noise when using a high temperature in the flow. A too low temperature would also impact the results by smoothing the image too much. As was shown in Section 10.1 and 9.2.

Multi-scale structure:
In practice we saw a trade-off between the ability of the model to reconstruct and the ability of the variational priors to follow the variational posterior given the L-levels of the flow. With the model being able to more easily reconstruct the image with a lower depth, but having a harder time making predictions. This behaviour is expected as the latent space will encode more high-level features with a high L. A reason to this is that the latent dimensions decrease based on L and thus also a greater potential for a better generalisation. On the other hand a latent space with too few dimensions, such as for a high L, might not approximate the underlying data distribution well enough. With a better generalisation the high-level features will be better approximated by the variational prior. In Glow [6] and VideoFlow [1], there was a sweet spot around level $L = \{3, 4\}$ for 64×64 images. In the RFN we noticed the same behaviour. It is therefore assumed that a very deep flow can lead to poorer reconstructability and a too shallow flow might lead to a poorer generalisation.

Computational cost and constraints:
Given the large computational cost of training normalising flow based models on images together with the temporal representation of video data. The problem of optimising the RFN model becomes a very computational demanding process. This combined with limited computational resources led to only the final RFN models could train until convergence. A proper hyper-parameter optimisation would require either more time or GPUs. Most of the settings for the final models are, therefore, approximated by other papers or by the limited results found for the different parameters in Section 9.1 and 9.2. It is also the reason that the regularisation methods were only used on the SRNN models, this allowed us to train the models until convergence. The results of the skip connections can, therefore, be slightly biased and might also explain the minor difference found in the validation losses. The temperature factor was evaluated on the final models, which satisfied the convergence criteria.

Another simplification that had to be made, was to only do optimisation of hyper-parameters on a single data set and then generalise the results to other data set. In an ideal situation this would have been done for each separate data set, as we did notice differences in optimal settings across data sets. The reason for not doing this was a limited amount of time.

The Glow is a very memory dependent model as the model utilised a large amount of parameters. Due to the limited amount of memory on the Tesla V100 GPUs the RFN was forced to train using a lower number of flow-steps, K, than what is otherwise typically seen used in papers [6, 41]. In most papers, the number of flow-steps range from 24 up to 32, while in our case only 10 were used for the fully trained models. An improvement might have been achieved if K was higher, as a higher K allows for more expansions and contractions of the initial distribution leading to a more multi-modal and complex density estimation [70]. To make training with a higher K, gradient checkpointing may be used as suggested in Sohoni et al. [71]. Gradient checkpointing works by storing only a subset of the network activation. Consequentially, the activation not stored must be recomputed during a backward pass. This makes each pass slower because of the extra computation. The procedure is still numerically unchanged leaving the accuracy unaffected. The benefit of gradient checkpointing is that we noticed a huge reduction in memory, sometimes up to a 15 GB, with a moderate impact on computation time. The huge reduction in memory was from not storing all activation used in the affine coupling step. Gradient checkpointing was not included for the results as the idea of gradient checkpointing was only discovered during the final stage of this thesis.

Training Difficulties:
Training the RFN model can be difficult. We noticed divergence could spontaneously occur without the right tweaks and settings. We argue the divergence is a two part problem. First part being an effect of a narrow parameter gradient manifold inherent from the combination of different model architectures when unifying the SRNN and

the CNF. It might have been possible to alleviate this problem with specialised optimisers for the different parts of the model [68]. Second part of the problem being the training instabilities introduced by modelling variability as with other auto-encoders. To alleviate the problems, a fairly low learning rate and β annealing was used. It was also necessary to further constrain the variance in the base distribution of the flow by using a `Softplus` activation function instead of the usual exponential. In general the flow is very sensitive to initialisation and much time went into properly initialising the flow and making sure the bijective property was fully satisfied[1].

Evaluation Methods:
As seen in Figure A.13 and A.14, the evaluation methods, such as SSIM, PSNR, and LPIPS, are quite biased towards the background. Often the background was estimated exceptionally well when modelling the data for the RFN, as the non-probabilistic evaluation methods emphasise the background by giving it a high score. Given the probabilistic nature of the model, when plausible prediction outcomes do not line-up with the ground truth sequence, the conventional metrics will heavily penalise. To alleviate the problem the scores are sampled several times. The sampling might make the predictions line up at some point with the ground truth due to the stochasticity of the predictions. However, we still notice that the best samples are still heavily dominated by images emphasising the background and the worst by more stochastic images. This is probably due to a mix of the model not performing optimally and the stochastic behaviour of long-range generations.

In practice, we noticed that the SSIM, PSNR, and LPIPS were not very good measures to rely on. This is also recognised in newer literature where a paradigm shift away from the conventional metrics and towards the more recently proposed FVD metric is happening [67]. The problem with FVD is that it is expensive to compute, however the qualitative metric introduces some appealing properties, such as sensitivity to visual quality, temporal coherence and diversity of samples [67]. It is shown to correlate more with human perception [1, 67]. In this thesis it was also noticed that the conventional metrics did not always correlate with perceptual quality. As such, in the ideal case a more thorough investigation of the hyper-parameters and their effects would have been conducted using a combination of the conventional and FVD metrics.

Benefits and disadvantages:
The main advantage of the RFN model is that it is possible to get a more exact measure of the output distribution $p(\mathbf{x}|\mathbf{z}, \mathbf{h})$. It improves the modelling of variability in the data. The SRNN and VRNN also model the exact log-likelihood of the output distribution, but it lacks the flexibility of flows when using a discretized mixture of logistics. The flexible approximation of flows is useful when the model needs to

[1] To check the bijective property, an input is given to the normalising direction $\mathbf{z} = f(\mathbf{x})$ and then $\mathbf{z}$ is given to the generative direction $\mathbf{x} = f(\mathbf{z})^{-1}$. The flow should then be able to reconstruct a copy of the image x.

adapt to the data. This can reduce the distance between the underlying true output distribution and the model distribution. Furthermore, normalising flows are known to have fast and efficient sampling from their bijective nature [6, 36]. As such, adding a Glow to the RFN should not limit the efficiency, rather the recurrent nature of the RFN limits the efficiency of the models.

The added flexibility of flows also comes with disadvantages. We noticed a very large increase in computational demand. Normalising flows are difficult to train for high-dimensional data, requiring deep networks, careful initialisation and a decent amount of tricks.

Artifacts:
In some cases we experienced tiny checkerboard artifacts, in a recent paper from Lugmayr et al. [41] they found that artifacts appear because of the squeeze operation in the flow. The squeeze operation halves the spatial resolution each time it is called. This operation is only based on pixel reordering and can therefore introduce some strange behaviour when reconstructing the image. A possible solution would have been to introduce a transition step after each squeeze operation. The transition step removes the coupling operation for a few flow steps and only uses the invertible 1×1 convolution permutation along with an `ActNorm`. This gives the flow an opportunity to learn a linear invertible interpolation between neighbouring pixels [41].

Density estimation in video prediction:
Conventional video prediction algorithms do not model the conditional output probability $p(\mathbf{x}|\mathbf{z})$, but rather estimate it with a Gaussian distribution [60, 66] or not at all [52]. In this thesis, the models generate results by using an exact conditional output probability. Density estimation of the conditional output probability is a challenging problem, as the network has to model the underlying variability. Given the majority of models found in literature [7, 52, 57, 65, 66, 68] do not estimate the exact variability of the data this should be taken into consideration when a comparison is conducted against exact likelihood models.

Conditioning network:
For the adaption of the RFN to the video generation domain, several design choices were made to ensure stability and convergence. One design choice was to upsample the recurrent latent features to fit to the multi-scale structure of the conditional normalising flow. This was done to match spatial sizes of the recurrent conditions $\mathbf{z}_t, \mathbf{h}_t$ to every L level of the CNF, needed for the CNF to function. Adding this conditioning network to upsample the recurrent features will mean that the possibility of the flow receiving the same information multiple times is possible. This adds redundancy in the network and extra parameters, as the CNF also upsample the input through the L levels. We experimented with other architectures for conditioning the CNF, trying to avoid the upsampling of the recurrent conditions. However, in the end the presented architecture allowed for convergence and stable training of the model. To

mention some of the experiments we tried, but with no success; directly reshaping the conditions to match the CNF by a small network, having recurrent latent features for every L-level of the CNF, and reshaping and upsampling $\mathbf{z}_t, \mathbf{h}_t$ for every L-level. None of these approaches were optimal.

Consistency:
In many of the SSIM/PSNR/LPIPS figures it is seen that the one-step prediction performs much better than the future predictions. A possible explanation to this is first of all that the one-step prediction is conditioned on the last given ground truth frame. However, the decrease (increase, if LPIPS) does seem to be too excessive for the conditioning on the ground truth to be the only reason. We argue that a second reason could be caused by minute differences in the noise produced by the RFN as seen in Figure 11.1. Flexible models can be sensitive to noise [72] and added noise of the one step predictions might be causing the predictions of the model to deviate faster over a series of time steps.

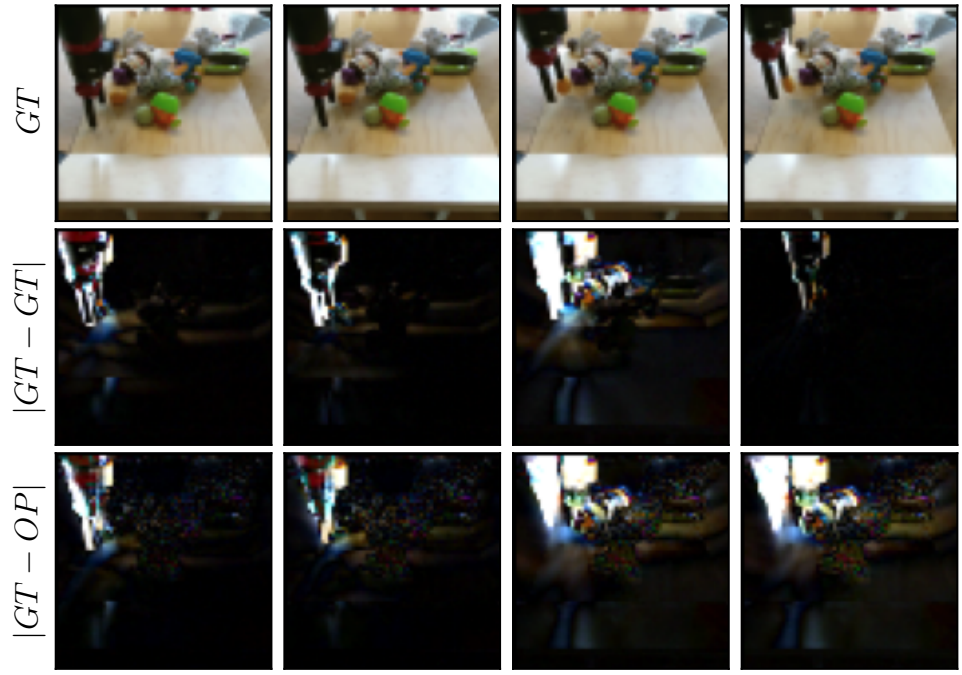


Figure 11.1: The noise level of a reference ground truth sequence GT, the associated difference between two ground truth images between time steps $|GT - GT|$, and the difference between a one step prediction OP of the RFN and the ground truth to the predicted time step $|GT - OP|$.

Training methods:
It was in general difficult to compare the RFN to other models from the literature. The reason being that there is no consensus on how to establish a common evaluation methodology. This means, that almost every model use a different approach to training or testing. The problematic of this is emphasised when training the model. Between studies it was observed that there was a wide variation in the number of frames seen during training. This made it difficult to draw a final conclusion on the overall performance of the RFN without training a new model for each unique case. However, the RFN in ideal situations and given its relative low number of seen frames when trained did seem to perform very well, such as was seen with the BAIR data set.

CHAPTER 12

Conclusion

This thesis presented the theory behind recurrent generative models. Specifically, an emphasis was placed on the stochastic recurrent neural network and the variational recurrent neural network. It was seen how it is possible to use variational theory to model temporal problems by introducing stochastic and deterministic latent states. In addition, the theory of normalising flows explained how it was possible to use bijective transformations, to transform a simple distribution to a highly complex non-Gaussian distribution through a series of invertible transformations. By extending the theory behind Glow it was shown that introducing conditional states to the affine coupling layers and base distribution makes it possible to model complex conditional distributions with the useful properties of a flow: efficient sampling and exact log-likelihood. Together, the theory of recurrent generative variational models and conditional flows are unified, solidifying the foundation for the Recurrent Flow Network. Different regularisation methods, not previously used in a video setting, were investigated on the SM-MNIST data set. No performance benefit was seen with latent overshooting, but a minor performance gain was achieved using smoothing or RES_q on the SRNN for video generation.

By implementing a unique architecture a generalisation of the RFN to video data was possible. The RFN was able to estimate the underlying high-dimensional distribution and generate realistic predictions of video data. A key point of this thesis was the focus on properly estimating the variability in data. As such, an ablation study of the RFN and its properties was conducted. An interesting behaviour of the stochastic dynamics was found by examining the temperature factor in the stochastic variational distributions and the flow base distributions. It was shown that the RFN exhibits disentanglement in the temporal and spatial stochasticity. It enables an extra degree of freedom to adjust different stochastic behaviours in the model independently of each other.

Quantitative and qualitative results for a recurrent flow network were presented. For a more exact measure of bits-per-pixel a discretized mixture of logistics was used with the SRNN and VRNN. The RFN was compared against VRNN, SRNN, and other models found in current literature. It was discovered that the RFN performed better when trained on raw realistic RGB data, such as the BAIR data set, but it did struggle with grey-scaled data, such as SM-MNIST and KTH.

For the RFN trained on the SM-MNIST data set a significant decrease in performance was observed compared to its ancestor, the SRNN. The RFN was not able to model the overlap of two digits. To investigate the problem the variational and flow parameters were used to explain irregularities in the spatio-temporal dynamics. It was seen that the model mistakenly models the overlap of digits as a statistical stochastic process. As the SRNN and VRNN did not have the same issue, the problem was narrowed down to issues with the conditional Glow architecture of the RFN. The Glow is known to have a general issue with modelling regions of low probability and disjoining distributions in the latent manifold. This is suspected to have manifested itself in the inability to model overlapping digits.

On the BAIR data set the performance of the RFN was found to be competitive. It did not outperform VideoFlow, but it was superior against VRNN and SRNN. The background and smaller details were modelled well by the deterministic latent states of the RFN, while diversity was also found to be high in the stochastic latent states. Even in long range predictions the RFN performed in a robust manner. We argued that the RFN prefer; movement that is uni-modal, i.e. only a single object is moving, and video with a rich dynamic range. The BAIR data is RGB data with a more continuous range of values compared to SM-MNIST, this allowed the Glow to have a smoother manifold without disjoint distributions.

Finally, the RFN was trained on the KTH data set. Its performance was on par with the SRNN and VRNN. As the test setup is not identical it was difficult to compare its performance against competing models. However, it did seem promising looking at the FVD score when taking into account the more limited total frames used for training the RFN. It was observed that the RFN struggled to generate high fidelity predictions, which lead to the loss of smaller details, such as hands and feet. We argued that this is a difficult case to model in general, but might be contributed to the difficulty of the Glow to model low probability edge cases.

To summarise, it was difficult to compare performances with different studies, as each study had a different training method. We did not have time to train a unique model for each case, instead it was simply taken into consideration when evaluating the model. However, a large potential in the RFN is recognised. The idea of variational density estimation with exact and flexible estimation of flows is appealing. The RFN performed well on the BAIR data set, but improvements could be made to the model for a better density estimation on complex dynamics, such as found in the SM-MNIST data set.

12.1 Future work

During the work of this thesis many suggestions inspired by the work of others, lead to interesting ideas for future work. In this section a few of the ideas are presented. A suggestion is to explore a hierarchical network structure as in Castrejon et al. [65]. Hierarchical structures have shown to give state-of-the-art results [65, 73, 74] by disentangling the latent space. This disentanglement comes from using multiple levels of latent variables. It allows each stochastic latent variable to describe different underlying features based on their spatial resolution of the data leading to a more flexible variational prior and posterior. By having a hierarchical structure it is possible to model the conditions for the l-levels of the different flow steps in the flow more directly, without having to up-sample the latent features. A challenge in this will be to optimise the model, as hierarchical structures are difficult to train and the Glow can also exhibit unstable and irregular behaviour.

Often a context variable is implemented into video prediction models. A context variable is a variable encoding static information of the sequence. It allows the model to focus more on the temporal aspects of the video generation. It has shown to improve results and could easily be added to the RFN model [60, 65]. It is noticed that for long term predictions spatial features morph/distort from the ground truth reference, a context variable might help alleviate this problem. Furthermore, the checkerboard artefacts that appear in the predictions, especially for the BAIR data set, might have been resolved by using a transition step, as is done in SRFlow [41].

Training the RFN model was a computational demanding and slow process. To improve training Lugmayr et al. [41] inserted a conditional affine injector between flow layers. By using a conditional affine injector they achieved significant faster convergence, as well as a higher effectiveness of the conditional variables. Inserting this affine injector is expected to grant the same benefits to the RFN.

A number of recently introduced flow methods beat the performance of the Glow model. One could take advantage of newer flow structures to see an overall improvement in the RFN. Ideas such as Flow++ by Ho et al. [75] extends and improves the overall structure of the Glow and could significantly improve the performance of the RFN.

A final suggestion for improvement could be to use a mix of planar and affine coupling layers as was presented in Tygesen [37]. Here it was discovered that by mixing coupling layers and planar flows, it was possible to alleviate the problem of disjoint distributions and low probability edge cases. We reckon this might also be the case for the conditional Glow and could improve the modelling of the underlying data manifold.

With the above suggestions we expect the RFN to see significant improvements.

Bibliography

[1] Manoj Kumar et al. *VideoFlow: A Conditional Flow-Based Model for Stochastic Video Generation*. 2020. arXiv: 1903.01434 [cs.CV].

[2] Marco Fraccaro. "Deep Latent Variable Models for Sequential Data." PhD thesis. Technical University of Denmark., 2018.

[3] Marco Fraccaro et al. *Sequential Neural Models with Stochastic Layers*. 2016. arXiv: 1605.07571 [stat.ML].

[4] Chelsea Finn, Ian Goodfellow, and Sergey Levine. *Unsupervised Learning for Physical Interaction through Video Prediction*. 2016. arXiv: 1605.07157 [cs.LG].

[5] Daniele Gammelli and Filipe Rodrigues. *Recurrent Flow Networks: A Recurrent Latent Variable Model for Spatio-Temporal Density Modelling*. 2020. arXiv: 2006.05256 [stat.ML].

[6] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. 2018. arXiv: 1807.03039 [stat.ML].

[7] Christina Winkler et al. *Learning Likelihoods with Conditional Normalizing Flows*. 2019. arXiv: 1912.00042 [cs.LG].

[8] Gerben van den Broeke. "What auto-encoders could learn from brains Generation as feedback in unsupervised deep learning and inference." In: 2016.

[9] Brendan van Rooyen and Robert C. Williamson. *A Theory of Feature Learning*. 2015. arXiv: 1504.00083 [stat.ML].

[10] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. *Testing the Manifold Hypothesis*. 2013. arXiv: 1310.0425 [math.ST].

[11] Geoffrey E. Hinton. "To recognize shapes, first learn to generate images." In: *Progress in Brain Research, volume 165* (2007), pages 535–547. ISSN: 0079-6123. DOI: https://doi.org/10.1016/S0079-6123(06)65034-6.

[12] Geoffrey Hinton. "Learning multiple layers of representation." In: *Trends in cognitive sciences* 11 (November 2007), pages 428–34. DOI: 10.1016/j.tics.2007.09.004.

[13] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE." In: *Journal of Machine Learning Research* 9 (2008), pages 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[14] Harri Valpola. *From neural PCA to deep unsupervised learning*. 2015. arXiv: 1411.7783 [stat.ML].

[15] Xingjian Shi et al. *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*. 2015. arXiv: 1506.04214 [cs.CV].

[16] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders." In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pages 307–392. ISSN: 1935-8245. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[17] Carl Doersch. *Tutorial on Variational Autoencoders*. 2016. arXiv: 1606.05908 [stat.ML].

[18] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[19] S. Gershman and Noah D. Goodman. "Amortized Inference in Probabilistic Reasoning." In: *Cognitive Science* 36 (2014).

[20] Gregory Gundersen. *The Reparameterization Trick*. 2018. URL: http://gregorygundersen.com/blog/2018/04/29/reparameterization/.

[21] Najmeh Abiri and Mattias Ohlsson. *Variational auto-encoders with Student's t-prior*. 2020. arXiv: 2004.02581 [cs.LG].

[22] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. *Importance Weighted Autoencoders*. 2016. arXiv: 1509.00519 [cs.LG].

[23] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory." In: *Neural computation* 9 (December 1997), pages 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[24] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].

[25] Junyoung Chung et al. *A Recurrent Latent Variable Model for Sequential Data*. 2016. arXiv: 1506.02216 [cs.LG].

[26] Samuel R. Bowman et al. *Generating Sentences from a Continuous Space*. 2016. arXiv: 1511.06349 [cs.LG].

[27] Casper Sønderby et al. "How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks." In: (February 2016).

[28] Diederik P. Kingma et al. *Improving Variational Inference with Inverse Autoregressive Flow*. 2017. arXiv: 1606.04934 [cs.LG].

[29] Danijar Hafner et al. *Learning Latent Dynamics for Planning from Pixels*. 2019. arXiv: 1811.04551 [cs.LG].

[30] Zhifeng Kong and Kamalika Chaudhuri. "The Expressive Power of a Class of Normalizing Flow Models." In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Edited by Silvia Chiappa and Roberto Calandra. Volume 108. Proceedings of Machine Learning Research. Online: PMLR, 2020, pages 3599–3609. URL: http://proceedings.mlr.press/v108/kong20a.html.

[31] Xi Chen et al. *Variational Lossy Autoencoder*. 2017. arXiv: 1611.02731 [cs.LG].

[32] Aaron van den Oord et al. *Conditional Image Generation with PixelCNN Decoders*. 2016. arXiv: 1606.05328 [cs.CV].

[33] Tim Salimans et al. "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications." In: *CoRR* abs/1701.05517 (2017). arXiv: 1701.05517. URL: http://arxiv.org/abs/1701.05517.

[34] George Papamakarios et al. *Normalizing Flows for Probabilistic Modeling and Inference*. 2019. arXiv: 1912.02762 [stat.ML].

[35] Ivan Kobyzev, Simon Prince, and Marcus Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pages 1–1. ISSN: 1939-3539. DOI: 10.1109/tpami.2020.2992934. URL: http://dx.doi.org/10.1109/TPAMI.2020.2992934.

[36] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. *Density estimation using Real NVP*. 2016. arXiv: 1605.08803 [cs.LG].

[37] Mathias Niemann Tygesen. *Density Estimation with Normalizing Flows*. 16. June 2020. URL: https://raw.githubusercontent.com/MathiasNT/DEwNF/master/Thesis/Density_estimation_with_Normalizing_Flows_thesis.pdf.

[38] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[39] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[40] Zhifeng Kong and Kamalika Chaudhuri. *The Expressive Power of a Class of Normalizing Flow Models*. 2020. arXiv: 2006.00392 [cs.LG].

[41] Andreas Lugmayr et al. *SRFlow: Learning the Super-Resolution Space with Normalizing Flow*. 2020. arXiv: 2006.14200 [cs.CV].

[42] Sergiu Oprea et al. *A Review on Deep Learning Techniques for Video Prediction*. 2020. arXiv: 2004.05214 [cs.CV].

[43] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity." In: *IEEE Transactions on Image Processing* 13.4 (2004), pages 600–612. DOI: 10.1109/TIP.2003.819861.

[44] Richard Zhang et al. *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*. 2018. arXiv: 1801.03924 [cs.CV].

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Neural Information Processing Systems* 25 (January 2012). DOI: 10.1145/3065386.

[46] Yunjey Choi et al. *StarGAN v2: Diverse Image Synthesis for Multiple Domains*. 2020. arXiv: 1912.01865 [cs.CV].

[47] Thomas Unterthiner et al. *Towards Accurate Generative Models of Video: A New Metric and Challenges*. 2019. arXiv: 1812.01717 [cs.CV].

[48] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG].

[49] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[50] Peter Doyle. *Why maximize entropy?* 1982. URL: https://math.dartmouth.edu/~doyle/docs/whyme/whyme.pdf.

[51] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database." In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[52] Emily Denton and Rob Fergus. *Stochastic Video Generation with a Learned Prior*. 2018. arXiv: 1802.07687 [cs.CV].

[53] Frederik Ebert et al. *Self-Supervised Visual Planning with Temporal Skip Connections*. 2017. eprint: arXiv:1710.05268.

[54] C. Schüldt, I. Laptev, and B. Caputo. "Recognizing Human Actions: a Local SVM Approach." In: *Proc. Int. Conf. Pattern Recognition (ICPR'04)*. Cambridge, U.K, 2004.

[55] Barbara Caputo Ivan Laptev. *Recognition of human actions*. 2005. URL: https://www.csc.kth.se/cvap/actions/.

[56] K. Lertniphonphan, S. Aramvith, and T. H. Chalidabhongse. "Feature extraction for human action classification using adaptive key frame interval." In: *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*. 2014, pages 1–6. DOI: 10.1109/APSIPA.2014.7041766.

[57] Mohammad Babaeizadeh et al. *Stochastic Variational Video Prediction*. 2018. arXiv: 1710.11252 [cs.CV].

[58] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[59] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[60] Jean-Yves Franceschi et al. *Stochastic Latent Residual Video Prediction*. 2020. arXiv: 2002.09219 [cs.CV].

[61] Niki Parmar et al. *Image Transformer*. 2018. arXiv: 1802.05751 [cs.CV].

[62] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[63] I. Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." In: *ICLR*. 2017.

[64] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32*. Edited by H. Wallach et al. Curran Associates, Inc., 2019, pages 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[65] Lluis Castrejon, Nicolas Ballas, and Aaron Courville. *Improved Conditional VRNNs for Video Prediction*. 2019. arXiv: 1904.12165 [cs.CV].

[66] Alex X. Lee et al. "Stochastic Adversarial Video Prediction." In: *CoRR* abs/1804.01523 (2018). arXiv: 1804.01523. URL: http://arxiv.org/abs/1804.01523.

[67] Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. *Scaling Autoregressive Video Models*. 2020. arXiv: 1906.02634 [cs.CV].

[68] Mohammad Babaeizadeh et al. "Stochastic Variational Video Prediction." In: *CoRR* abs/1710.11252 (2017). arXiv: 1710.11252. URL: http://arxiv.org/abs/1710.11252.

[69] Will Grathwohl et al. "Scalable Reversible Generative Models with Free-form Continuous Dynamics." In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rJxgknCcK7.

[70] Danilo Jimenez Rezende and Shakir Mohamed. *Variational Inference with Normalizing Flows*. 2016. arXiv: 1505.05770 [stat.ML].

[71] Nimit Sharad Sohoni et al. *Low-Memory Neural Network Training: A Technical Report*. 2019. arXiv: 1904.10631 [cs.LG].

[72] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML].

[73] Lars Maaløe et al. *BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling*. 2019. arXiv: 1902.02102 [stat.ML].

[74] Arash Vahdat and Jan Kautz. *NVAE: A Deep Hierarchical Variational Autoencoder*. 2021. arXiv: 2007.03898 [stat.ML].

[75] Jonathan Ho et al. *Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design*. 2019. arXiv: 1902.00275 [cs.LG].

[76] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild." In: *Proceedings of International Conference on Computer Vision (ICCV)*. December 2015.

APPENDIX A

Appendix

A.1 VRNN - Derivation of ELBO

Based on the factorisation from the graphical representation in Figure 3.3 the marginal probability can be found by marginalising out $\mathbf{z}$ and $\mathbf{h}$ of the joint probability

$$\log p_\theta\left(\mathbf{x}_{\leq T} \mid \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right) = \log \int p_\theta\left(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right) \mathrm{d}\mathbf{z}_{\leq T} \mathrm{d}\mathbf{h}_{\leq T}. \tag{A.1}$$

Equation A.1 is expanded by the factorisation of the inference $\mathbf{q}$ and the Jensen's inequality the log term can be moved inside the expectation

$$\begin{aligned}
\log p_\theta\left(\mathbf{x}_{\leq T} \mid \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right) &= \log \int \frac{p_\theta\left(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right)}{q_\phi\left(\mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{x}_{\leq T}, \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right)} q_\phi\left(\mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{x}_{\leq T}, \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right) \mathrm{d}\mathbf{z}_{\leq T} \ \mathrm{d}\mathbf{d}_{\leq T} \\
&= \log \mathbb{E}_{q_\phi\left(\mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{x}_{\leq T}, \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right)}\left[\frac{p_\theta\left(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right)}{q_\phi\left(\mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{x}_{\leq T}, \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right)}\right] \\
&\geq \mathbb{E}_{q_\phi\left(\mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{x}_{\leq T}, \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right)}\left[\log \frac{p_\theta\left(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right)}{q_\phi\left(\mathbf{z}_{\leq T}, \mathbf{h}_{\leq T} \mid \mathbf{x}_{\leq T}, \mathbf{u}_{\leq T}, \mathbf{h}_0, \mathbf{z}_0\right)}\right] = \mathcal{F}_i(\theta, \phi).
\end{aligned} \tag{A.2}$$

It is possible to remove $\boldsymbol{h}_t$ from the joint probability as it is deterministic

$$\mathcal{F}_i(\theta, \phi) = \mathbb{E}_{q_\phi\left(\mathbf{z}_{\leq T} \mid \mathbf{x}_{\leq T}, \mathbf{h}_{\leq T}\right)}\left[\sum_{t=1}^{T} \log \frac{p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right) p_\theta\left(\mathbf{z}_t \mid \mathbf{h}_t\right))}{q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{h}_t\right)}\right]. \tag{A.3}$$

Simplifying based on the log-term

$$\mathcal{F}_i(\theta, \phi) = \sum_{t=1}^{T} \mathbb{E}_{q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{h}_t\right)}\left[\log p_\theta(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t) + \log p_\theta\left(\mathbf{z}_t \mid \mathbf{h}_t\right) - \log q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{h}_t\right)\right]. \tag{A.4}$$

Finally, from the definition of the KL divergence the above equation can be written as

$$\mathcal{F}_i(\theta, \phi) = \sum_{t=1}^{T}\left[\mathbb{E}_{q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{h}_t\right)}(\log p_\theta(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t)) - \mathrm{D}_{\mathrm{KL}}(q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{h}_t\right) | p_\theta\left(\mathbf{z}_t \mid \mathbf{h}_t\right))\right], \tag{A.5}$$

which is the ELBO that should be optimized.

A.2 RFN - Derivation of ELBO

The derivation of the ELBO for the RFN follows the same procedure as in Appendix A.1.

$$
\begin{aligned}
\log p_\theta\left(\mathbf{x}_{\leq T}\right) &= \log \int p_\theta\left(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}, \mathbf{h}_{\leq T}\right) \mathrm{d}\mathbf{z}\mathrm{d}\mathbf{h} \\
&= \log \int \frac{q_\phi\left(\mathbf{z}_{\leq T} \mid \mathbf{x}_{\leq T}\right)}{q_\phi\left(\mathbf{z}_{\leq T} \mid \mathbf{x}_{\leq T}\right)} p_\theta\left(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}, \mathbf{h}_{\leq T}\right) \mathrm{d}\mathbf{z}\mathrm{d}\mathbf{h} \\
&= \log \mathbb{E}_{q_\phi\left(\mathbf{z}_{\leq T} \mid \mathbf{x}_{\leq T}\right)}\left[\prod_{t=1}^{T} \frac{p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right) p_\theta\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t\right) p_\theta\left(\mathbf{h}_t \mid \mathbf{h}_{t-1}, \mathbf{u}_t\right)}{q_\phi\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t\right)}\right] \\
&\geq \mathbb{E}_{q_\phi\left(\mathbf{z}_{\leq T} \mid \mathbf{x}_{\leq T}\right)}\left[\sum_{t=1}^{T} \log p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right) + \log p_\theta\left(\mathbf{h}_t \mid \mathbf{h}_{t-1}, \mathbf{u}_t\right) + \log\left(\frac{p_\theta\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t\right)}{q_\phi\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t\right)}\right)\right] \\
&= \sum_{t=1}^{T}\Big(\mathbb{E}_{q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t\right)}\left[\log p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right) + \log p_\theta\left(\mathbf{h}_t \mid \mathbf{h}_{t-1}, \mathbf{u}_t\right)\right] \\
&\quad - \mathrm{D}_{\mathrm{KL}}\left[q_\phi\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t\right) \| p_\theta\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t\right)\right]\Big) \\
&= \mathcal{F}_i(\theta, \phi).
\end{aligned}
\tag{A.6}
$$

Again it is possible to avoid the distribution of $\mathbf{h}_t$ as it is deterministic, it is simply a constant, for training the following ELBO is optimised

$$
\begin{aligned}
\mathcal{F}_i(\theta, \phi) = \sum_{t=1}^{T}\Big(&\mathbb{E}_{q_\phi\left(\mathbf{z}_t \mid \mathbf{x}_t\right)}\left[\log p_\theta\left(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{h}_t\right)\right] \\
&- \mathrm{D}_{\mathrm{KL}}\left[q_\phi\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t, \mathbf{x}_t\right) \| p_\theta\left(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{h}_t\right)\right]\Big).
\end{aligned}
\tag{A.7}
$$

A.3 Sinusoidal data set

The sinusoidal data set is the simplest data set used in this thesis. It is mainly used as a toy data set and is generated by a sinusoidal process.

$$
y = \sin(Fx + 2\pi u) + \epsilon \tag{A.8}
$$

The frequency is user defined by F, with the amplitude being fixed to 1. The noise ϵ is sampled from a normal distribution $\epsilon \sim \mathcal{N}(0, \sigma)$ where σ is given and added to the sine function, furthermore to add more stochasticity a term of $2\pi u$ where $u \sim U(0, 1)$ is also added to the input x. An example of a sequence can be seen in Figure A.1.

To verify and control the implementation of the VRNN and SRNN model. These models are trained on the Sinus data. All of the networks will be weight normalised feed forward neural networks with 3 layers. The activation function is a leaky ReLU. The dimensions of $\mathbf{h}_t$ and $\mathbf{z}_t$ is set to 100 and 15, respectively.

The data is segmented into temporal patches of 100 data points, with 30 of these temporal patches in one sequence. The difference between each x values is set to be

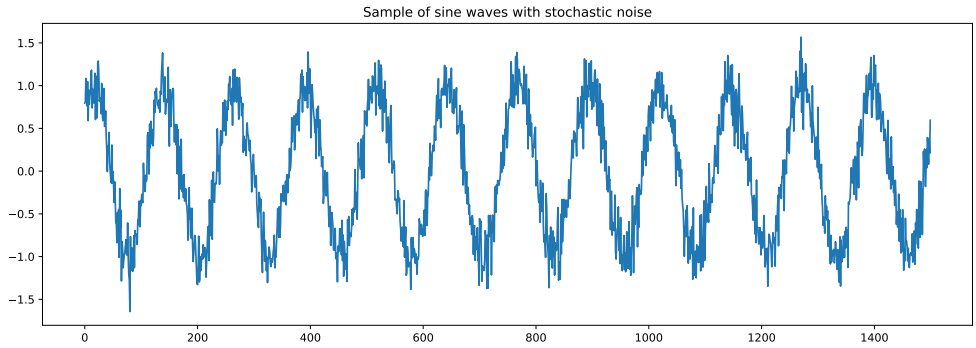


Figure A.1: A sequence of the sinusoidal data set.

0.05 and the noise value is set to 0.1.

The model is trained over the temporal patches, utilising an Adam optimizer [62] with a learning rate of 0.001. The learning rate is decreased when the loss function Equation 3.29 reaches a plateau.

All of the models are tested with or without the RES_q-algorithm, the smoothing step, and learnable initial parameters for $\mathbf{z}_0, \mathbf{h}_0$. The different test scores of the models are shown in Table A.1.

The best scoring model for SRNN and VRNN is shown in Figure A.2.

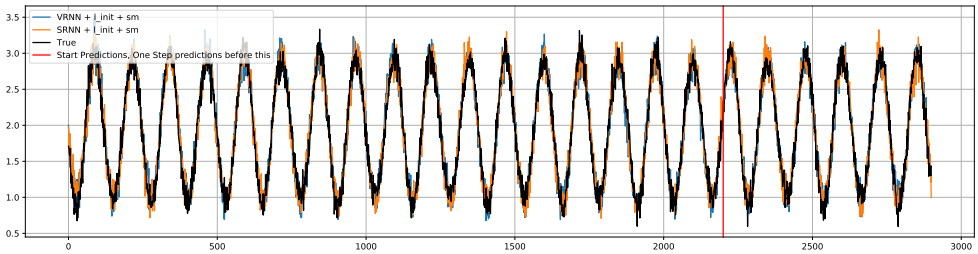


Figure A.2: SRNN and VRNN best performing models presented. The one with smoothing and learnable initial conditions seem to be superior.

VRNN	SRNN	RES_q	Learnable inits	Smoothing	Loss
✓					-1902.61
✓			✓		-1857.31
✓				✓	-1947.30
✓			✓	✓	-1953.76
✓		✓			-1880.57
✓		✓	✓		-1875.43
✓		✓		✓	-1945.33
✓		✓	✓	✓	-1913.03
	✓				-1866.13
	✓		✓		-1827.88
	✓			✓	-1902.55
	✓		✓	✓	-1960.56
	✓	✓			-1883.76
	✓	✓	✓		-1880.80
	✓	✓		✓	-1940.47
	✓	✓	✓	✓	-1943.24

Table A.1: Table of trained models, separated between SRNN and VRNN. The loss is for the validation set, lower is better. The best scenario for both models are underlined.

A.4 Quantized CelebA

Quantized CelebA is a data set that was used to test the conditioning of the normalizing flow. It is a reduced data set based on the CelebA [76] that is pictures of celebrity faces. It contains 26838 images, with 20000 used for training and 6838 used for testing. The images are 32×32 with 3 channels, each channel is 2 bits, hence the quantization, that means each pixel value will range from 0 to 3. Samples can be seen in Figure A.3. To make the data set eligible for conditional purposes we modify each images. We separate the image into a target and a condition. The target will be an inner square that is 16×16 of the image, the rest of the image is then the condition as seen in Figure A.4.

Figure A.3: 25 samples of the quantized CelebA

Figure A.4: A conditioned sample of the quantized CelebA. To the left is the condition, middle is the target and to the right the full image.

Albeit, this is not a temporal problem and can not be fully compared to the Glow in the RFN. The idea of using this data set was simply to verify that the conditional Glow model was able to model a conditioned problem and use the features from the conditioner. To do this the conditional Glow implemented follows Winkler et al. [7] and the theory in Section 4. The learning rate was 0.0001 and $K = 15$ with $L = 4$ and trained for 40 epoch. Uniform binning with 2 bits of noise was used. The data was preprocessed following the same procedure as in [6, 36]. In Figure A.5 some of the sampled faces can be seen. The condition used is taken from the test set.

Figure A.5: Generated faces from a conditional Glow model. The red border indicates the area of a generated face. Everything outside the border is the condition.

Only quantitative results are shown for this data set. It can be seen that facial features have been generated. The Glow also captures the border behaviour and skin colour fairly well. There is some blur, but this is expected given the data is only 2 bits. The uniform binning is also quite aggressive when the image is quantized a lot. Assuming a longer training the results could be improved, but the point was to illustrate that the conditioning works for a single static image.

A.5 Hyperparameter tables

In Table A.2 and A.3 the hyperparameters of the RFN and SRNN can be seen, respectively. The VRNN uses the same parameter as the SRNN.

Hyperparameter	**Value**
Flow Levels	4
Flow steps per level	10
Coupling type	Affine
Channels in coupling layer	256
Optimizer	Adam
Batch size	35
Learning Rate	0.0003
Training Steps	300000 steps
Channels in $\mathbf{z}_t$	3-5
Channels of $\mathbf{h}_t$	256
Channels of $\mathbf{a}_t$	256
Channels in base distribution	512
Flow normalisation	ActNorm
Network normalisation	Batchnorm
Linear decay scheduler	final 150000 steps
β warm-up	0.000001 to 1
RES_q	None
Latent overshooting, D	1
Smoothing	Enabled
Network type	VGG

Table A.2: Hyper-parameters of the RFN. $\mathbf{z}_t$ is spatial and varies depending on the data set. 5 for BAIR, 3 for SM-MNIST, 3 for KTH.

Hyperparameter	**Value**
Optimizer	Adam
Batch size	100
Learning Rate	0.0003
Training Steps	300000 steps
Channels in $\mathbf{z}_t$	12-25
Channels of $\mathbf{h}_t$	256
Channels of $\mathbf{a}_t$	256
Network normalisation	Batchnorm
Linear decay scheduler	final 150000 steps
β warm-up	0.000001 to 1
RES_q	None
Latent overshooting, D	1
Smoothing	Enabled
Network type	VGG

Table A.3: Hyper-parameters of the SRNN and VRNN, except, the VRNN does not use smoothing. $\mathbf{z}_t$ varies depending on the data set. 25 for BAIR, 15 for SM-MNIST, 12 for KTH.

A.6 Additional experiments

The effect of scaling the variance term of the variational prior is shown for the Max. Min. similarity measures for the BAIR data set in figure A.6. No significant improvement was seen.

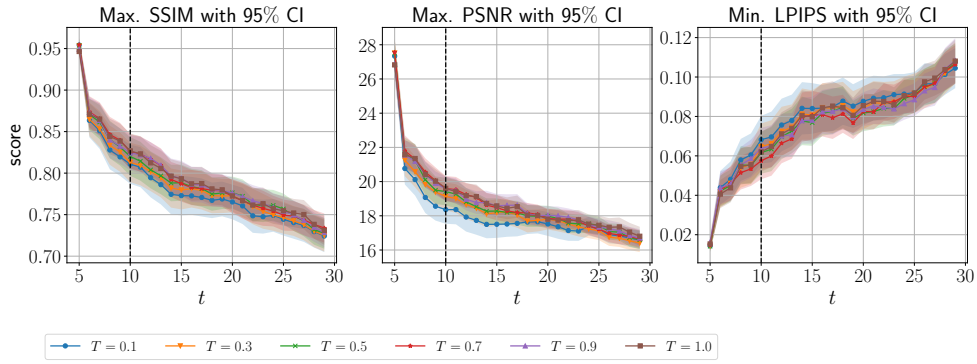


Figure A.6: The effect of scaling the variance term of the variational prior for the BAIR dataset.

A.7 Additional Analysis plots

In Figure A.8 and A.7 figures of the same type as in Figure 10.4 in Section 10.3 is seen for the BAIR and KTH data set.

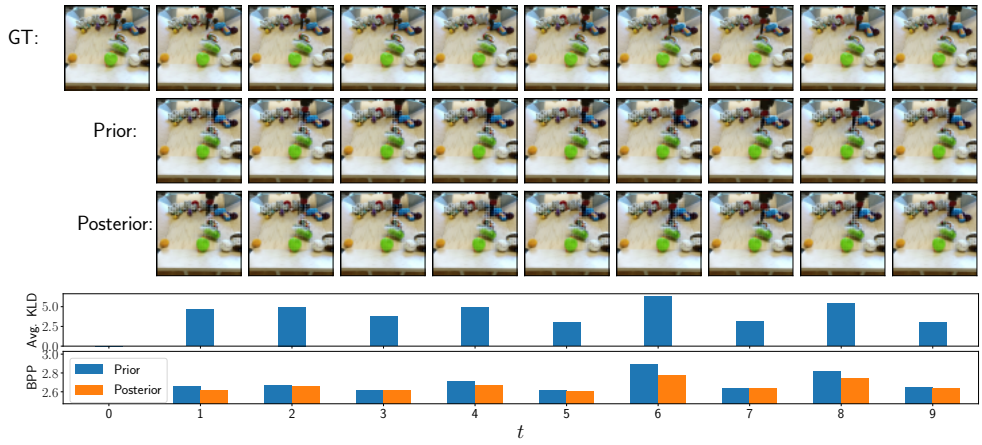


Figure A.7: Same Figure as in Figure 10.4 for the BAIR data set.

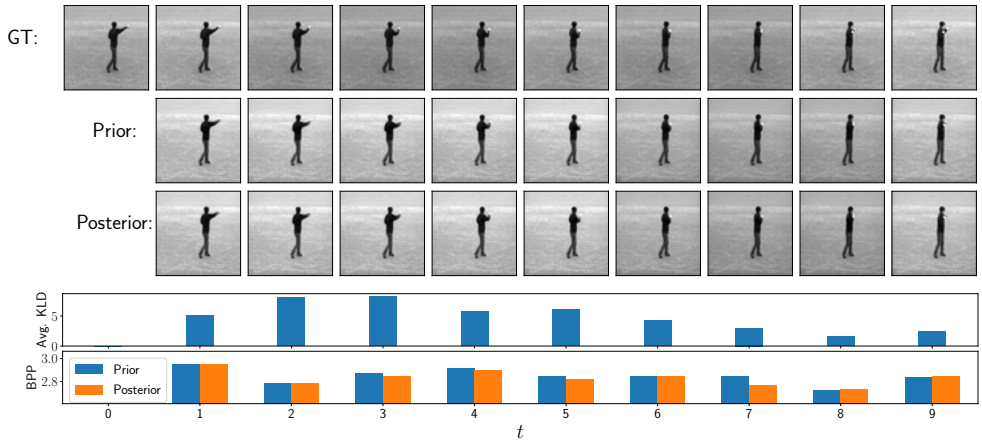


Figure A.8: Same Figure as in Figure 10.4 for the KTH data set

A.8 Additional SM-MNIST results

Diversity results of the SM-MNIST.

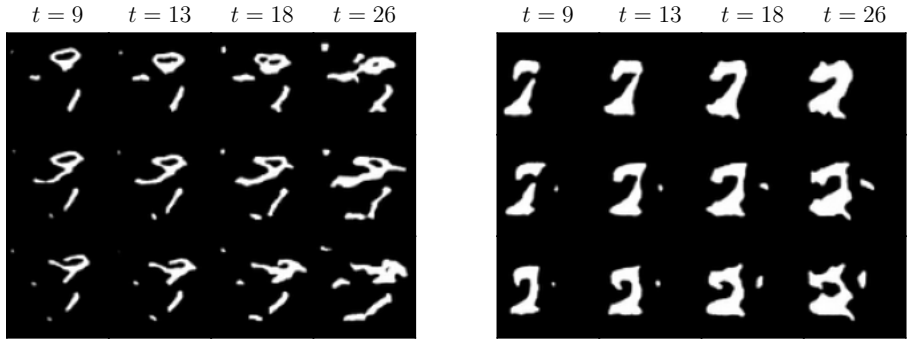


Figure A.9: Displaying three different predictions for two sets of conditioning frames (left and right). The figures showcase the diversity in outcomes. A temperature $T = 0.6$ is used.

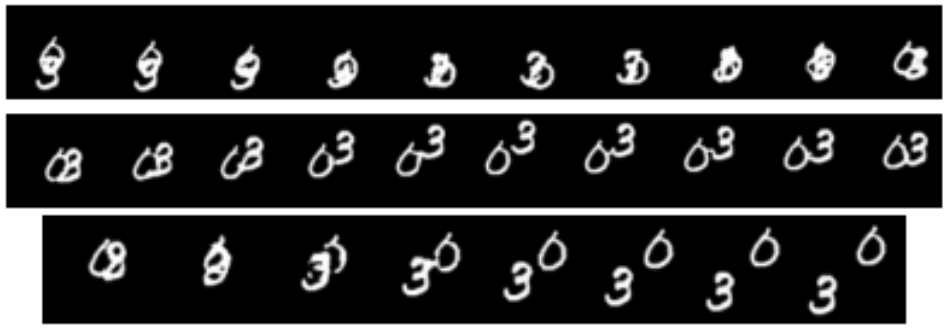


Figure A.10: The figure shows a single sequence of one step predictions used for the parameter behaviour in Figure 10.7

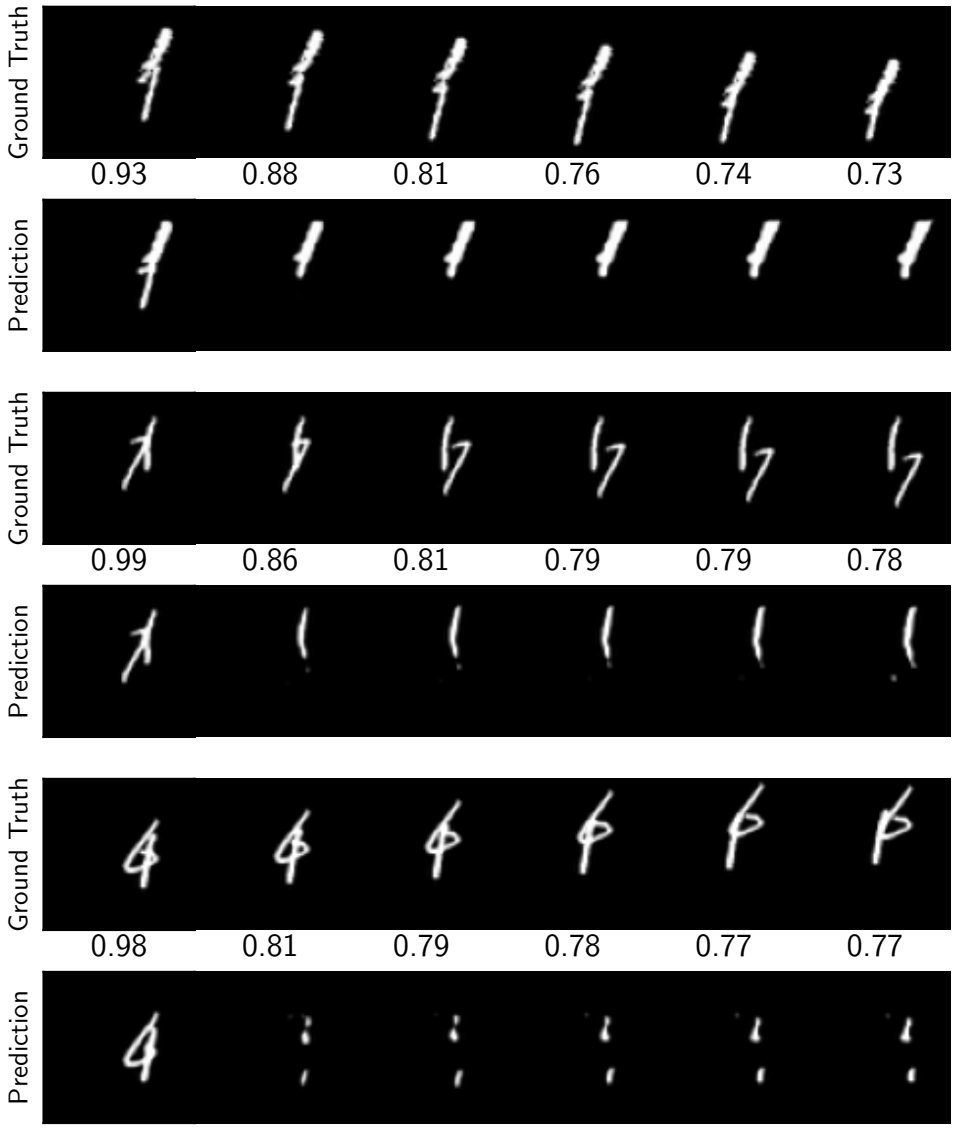


Figure A.11: Predictions of the Best samples based on SSIM for different video sequences. The value shown is the SSIM score between the ground truth and its corresponding prediction.

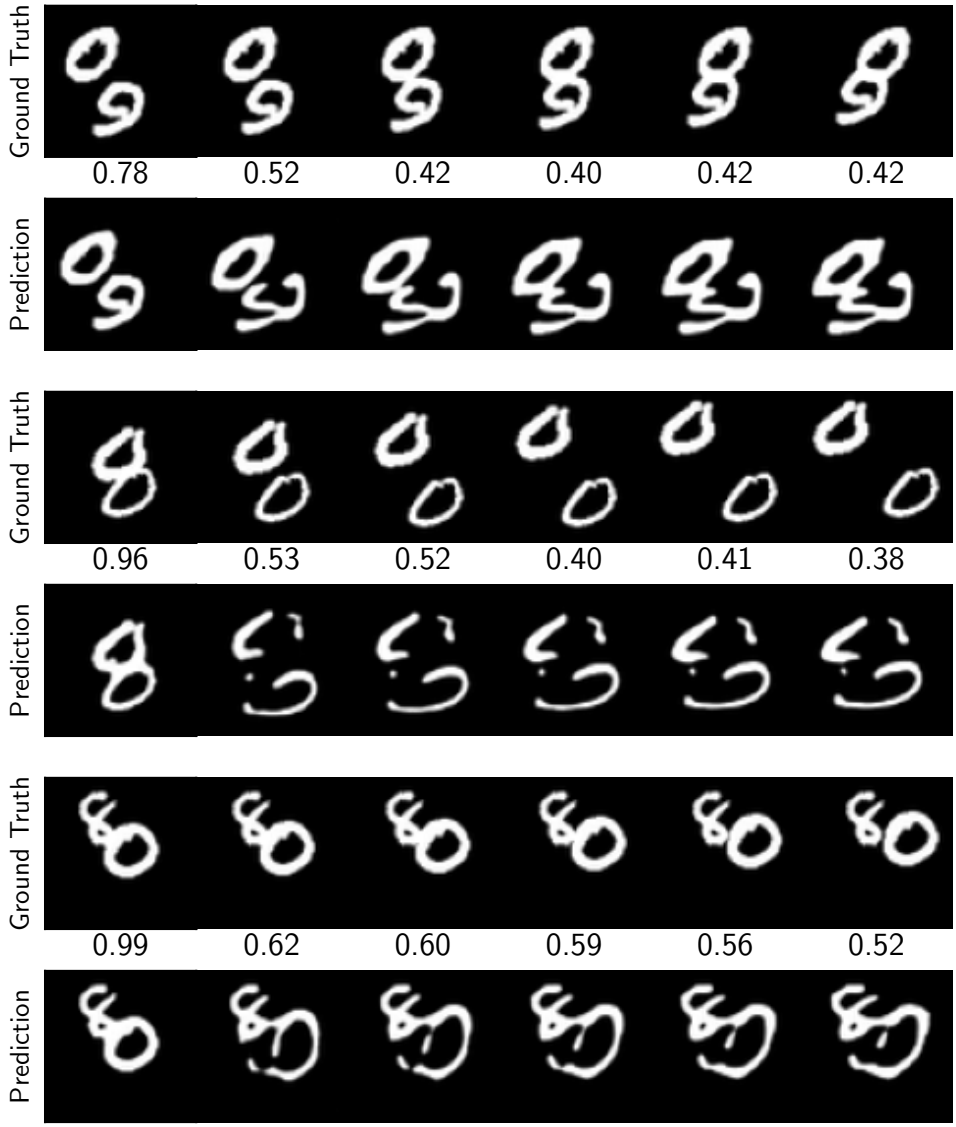


Figure A.12: Predictions of the worst samples based on SSIM for different video sequences. The value shown is the SSIM score between the ground truth and its corresponding prediction.

A.9 Additional BAIR results

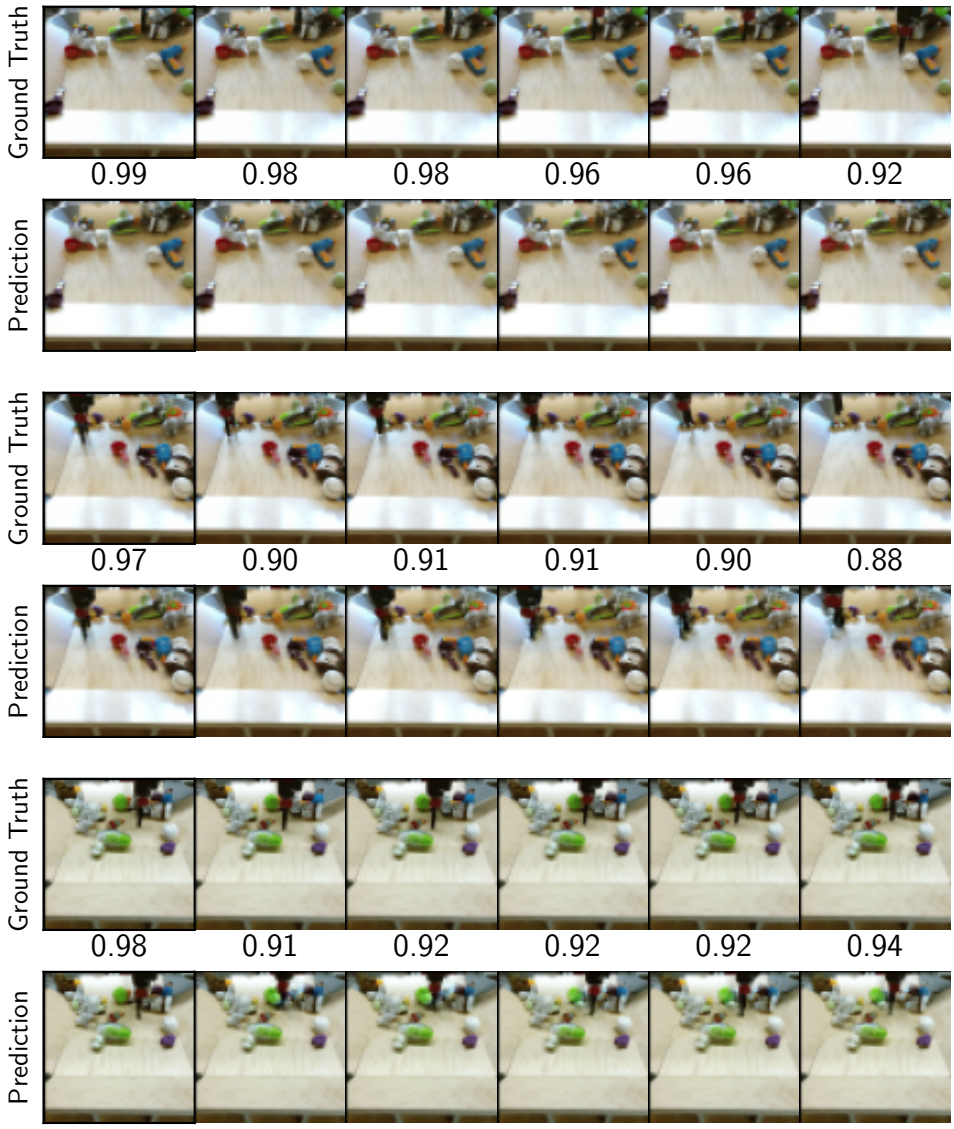


Figure A.13: Predictions of the best samples based on SSIM for different video sequences. The value shown is the SSIM score between the ground truth and its corresponding prediction.

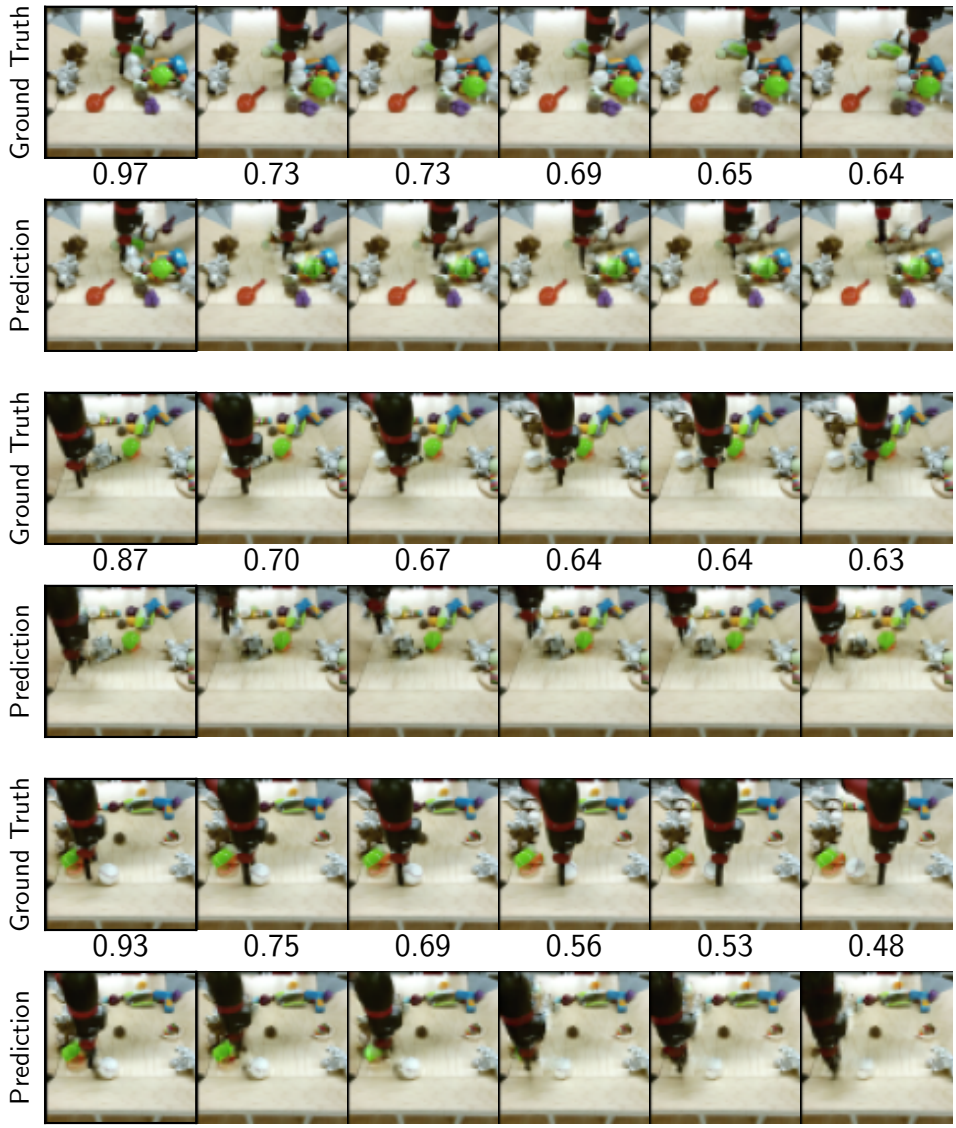


Figure A.14: Predictions of the worst samples based on SSIM for different video sequences. The value shown is the SSIM score between the ground truth and its corresponding prediction.

A.10 Additional KTH results

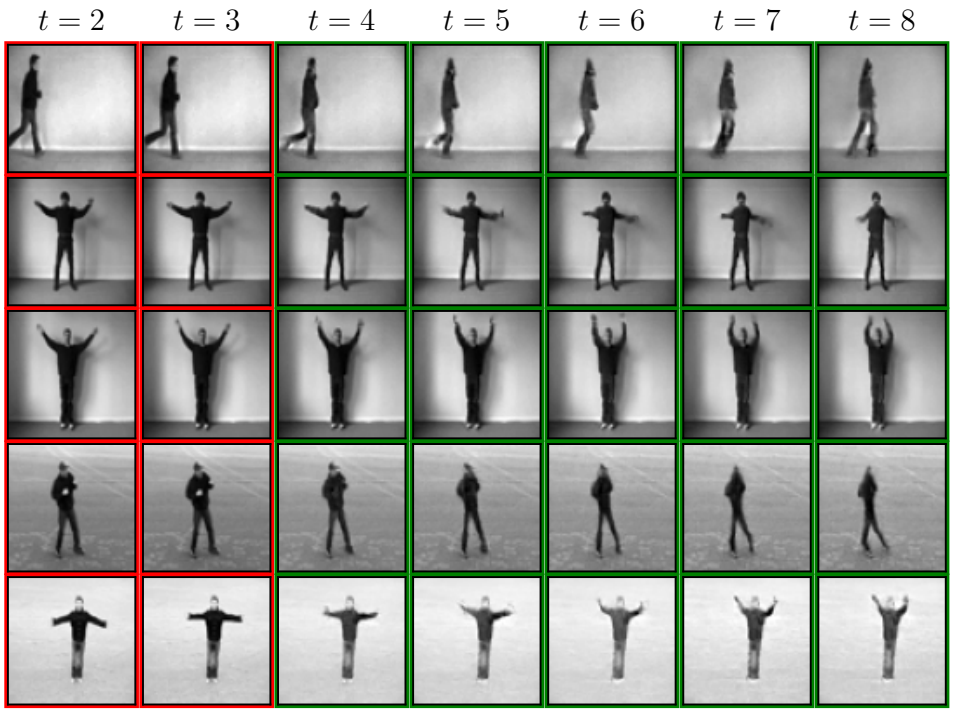


Figure A.15: 5 predictions conditioned (green) on 4 selected initial frames (red, only 2 shown) of different video sequences from the test set. The red border highlights a condition frame. The green highlights a prediction conditioned on the previous frame.

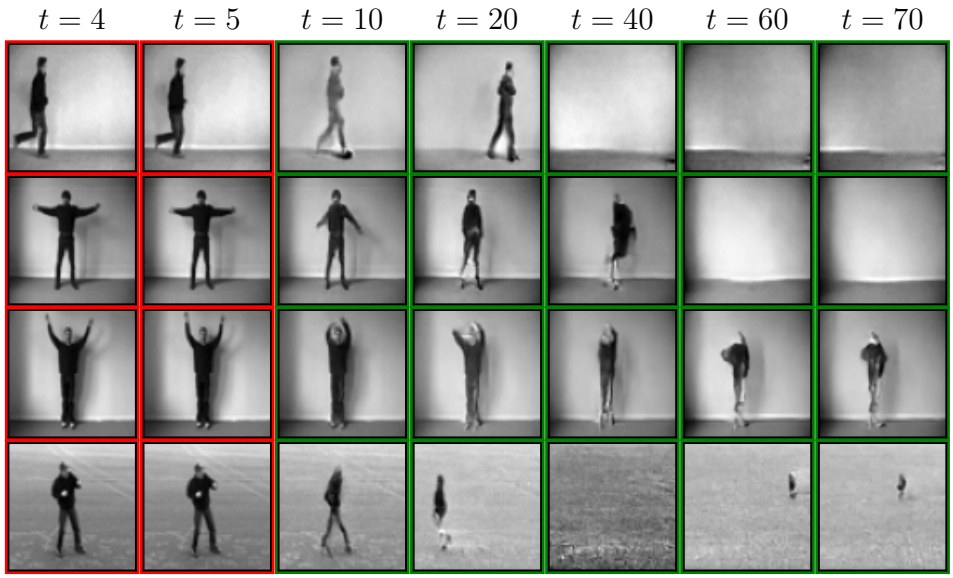


Figure A.16: Long range predictions of different sequences conditioned on the previous frame, warmed up on 5 frames. Red highlights the last 2 initial conditioned frames. Green highlights the predictions.

A.11 SRNN Results

In this section a few predictions of the SRNN is shown. It is seen that they do seem more pixelated, this is due to the output distribution used, which finds a distribution over pixels. This gives an exact log-likelihood, the downside being that images might turn out pixelated, instead of blurry as seen in the Gaussian output distribution.

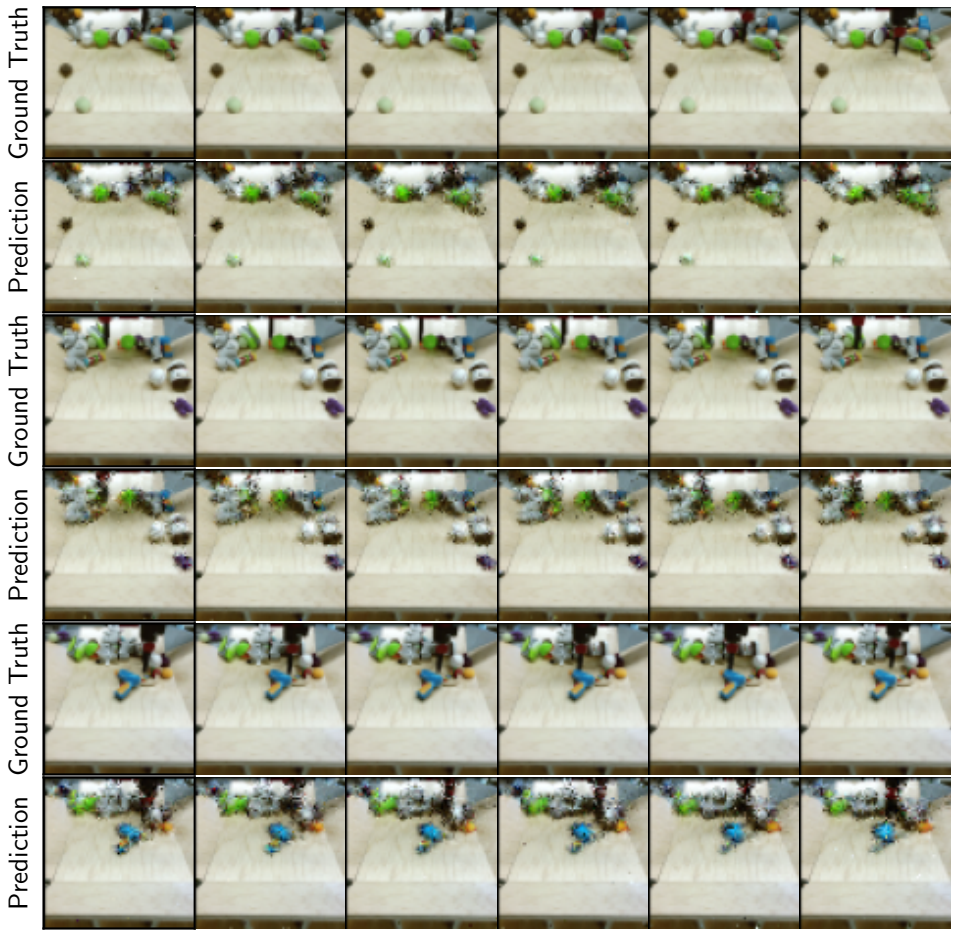


Figure A.17: Random predictions of SRNN on the BAIR data set. 3 frames used as warm-up.

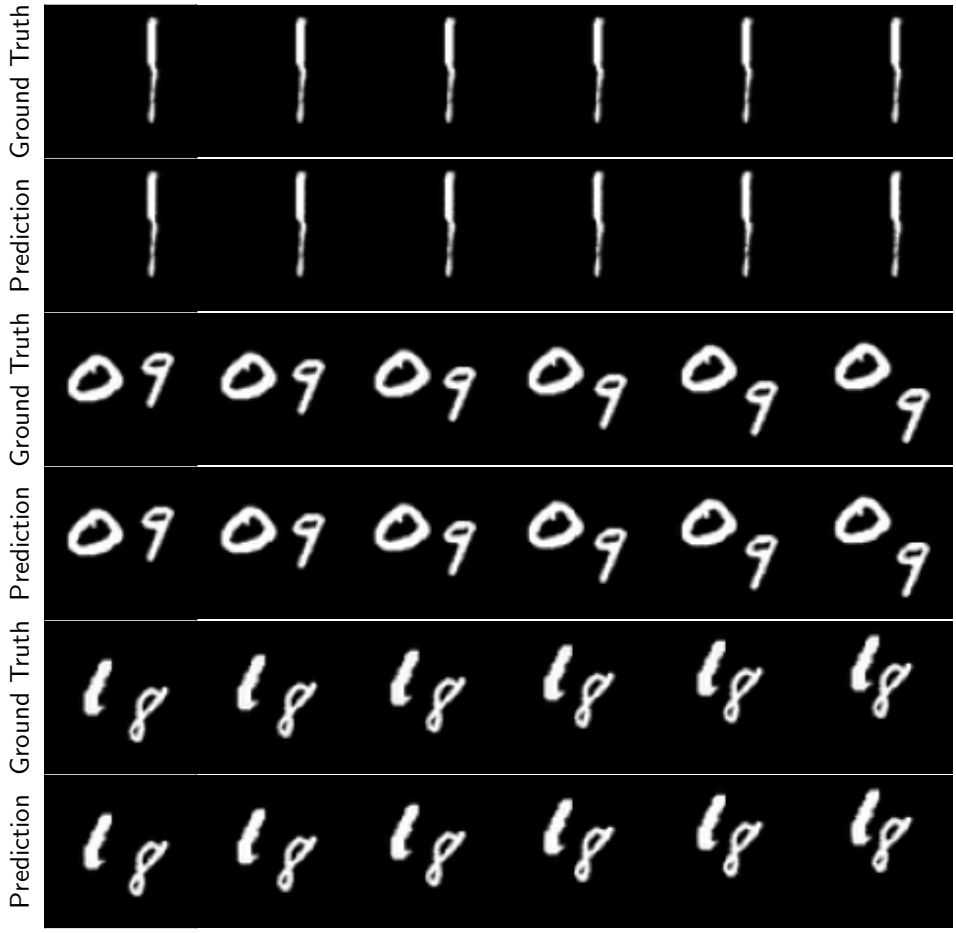


Figure A.18: Random predictions of SRNN on the SM-MNIST data set. 3 frames used as warm-up.

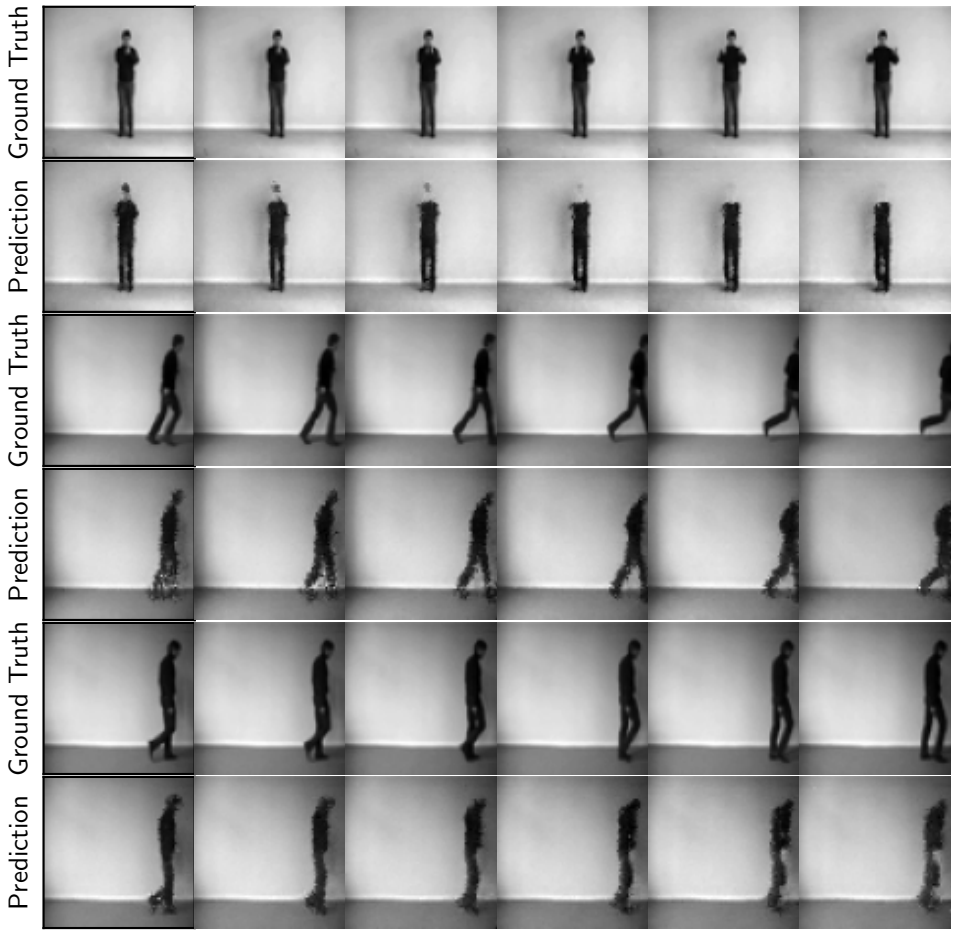


Figure A.19: Random predictions of SRNN on the KTH data set. 3 frames used as warm-up.

{2.7182818284}ϑφερτυθιοπσδφγηξκλ