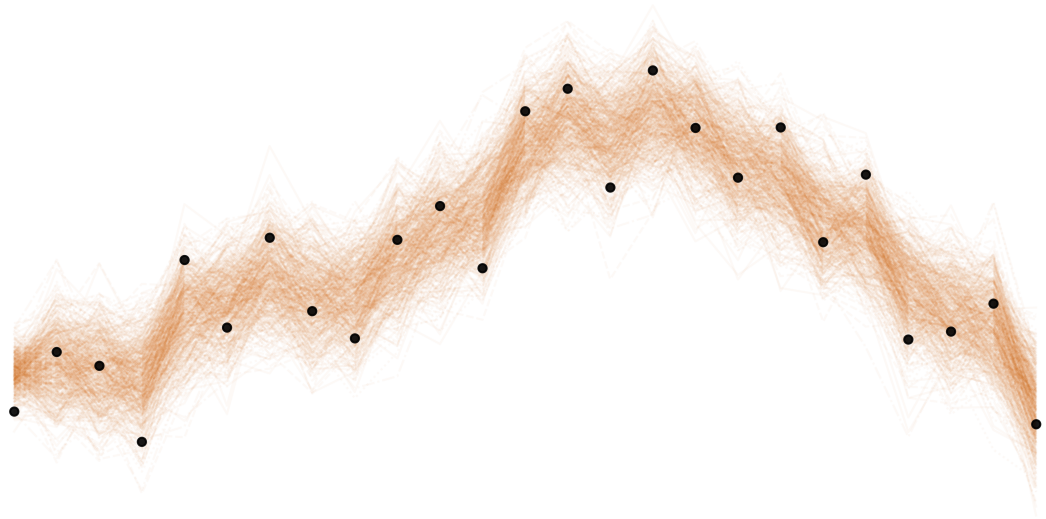


Sequential Normalising Flow for Approximating Smoothing Distributions

Evaluating Sequential Normalising Flow as a Cost-Efficient Alternative for Smoothing Distribution Estimation

Fredrik Boman

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2026

www.chalmers.se

Master's thesis 2026

Sequential Normalising Flow for Approximating Smoothing Distributions

Evaluating Sequential Normalising Flow as a Cost-Efficient Alternative for Smoothing Distribution Estimation

FREDRIK BOMAN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Science
Chalmers University of Technology
Gothenburg, Sweden 2026

Sequential Normalising Flow For Approximating Smoothing Distributions
Evaluating Sequential Normalising Flow as a Cost-Efficient Alternative for Smoothing Distribution Estimation
Fredrik Boman

Supervisor: Benjamin Svedung Wettervik, Saab
Supervisor: Karl Hammar, Saab
Examiner: Moritz Schauer, Chalmers Department of Mathematical Sciences

Master's Thesis 2026
Department of Mathematical Science
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The smoothing distribution approximated using the sequential normalising flow with 500 samples.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Sequential Normalising Flows for Approximating Smoothing Distributions
Evaluating Sequential Normalising Flows as a Cost-Efficient Alternative for Smoothing Distribution Estimation
Fredrik Boman
Department of Mathematical Science
Chalmers University of Technology

Abstract

This thesis addresses the smoothing problem, in which the goal is to estimate a time series of latent variables from a corresponding time series of noisy observations. Traditional methods such as the Kalman Smoother and the Unscented Kalman Smoother can solve this problem, but rely on Gaussian assumptions that introduce bias when applied to nonlinear systems, motivating the development of faster and more flexible alternatives. Hence, the Sequential Normalising Flow (SNF) is created, a model that allows sampling from an approximation of the smoothing distribution rather than computing it analytically. The SNF consists of an embedding model, implemented as a Gated Recurrent Unit (GRU), which encodes future observations into a fixed-dimensional representation, and a generative model implemented as a normalising flow. While training requires substantial computational resources, inference is significantly cheaper. The SNF is evaluated on two cases: a linear Gaussian state-space model and a nonlinear Gaussian state-space model. In both cases, traditional methods serve as a reference point. For the linear case, the SNF approximates the smoothing distribution well, with computational times comparable to those of the Kalman Smoother. In the nonlinear case, the SNF has more difficulty accurately estimating the smoothing distribution, which is believed to stem from the embedding model's limited ability to represent the observations. The thesis concludes that the SNF shows promise for certain cases and identifies the key difficulties in scaling the approach to nonlinear and more complex problems.

Keywords: bayesian smoothing, variational inference, normalising flows, deep learning, sequential normalising flow

Acknowledgements

First and foremost, I would like to thank my supervisors, Benjamin Svedung Wettervik and Karl Hammar, for their support throughout this process. Your guidance and encouraging spirit have meant a great deal to me. I would also like to thank my examiner, Moritz Schauer, for making this thesis possible and for many wonderful conversations. Additionally, I want to thank my opponents, Ellen Carlsson and Joakim Colpier, for their valuable input and for making my time at the office very enjoyable.

Lastly, I want to thank my partner, Helena Thim, whose support throughout this project has been invaluable. Without her, I do not think this would have been possible.

Fredrik Boman, Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ELBO	Evidence Lower Bound
FCNN	Fully Connected Neural Network
GRU	Gated Recurrent Unit
KF	Kalman Filter
KL	Kullback–Leibler
KS	Kalman Smoother
MSE	Mean Squared Error
NF	Normalising Flow
RNN	Recurrent Neural Network
SNF	Sequential Normalising Flow
SSM	State-Space Model
t-SNE	t-Distributed Stochastic Neighbour Embedding
UKF	Unscented Kalman Filter
UKS	Unscented Kalman Smoother
VI	Variational Inference

Contents

List of Figures

List of Tables

1
Introduction

The ability to detect and track unknown objects is a critical component of modern defence systems, providing situational awareness and enabling timely decision-making. A fundamental building block of such systems is radar, which detects airborne objects by analysing reflections of emitted radio waves. These reflections give rise to measurements that carry information about the object's state, such as its position and velocity. However, the relationship between the true state and the observed measurement is corrupted by noise, meaning the measurement is not an exact representation of the state. A principled framework for estimating the hidden state from noisy measurements is Bayesian inference, which represents uncertainty as a probability distribution. Given a probabilistic model of the state's dynamics and the measurement process, Bayesian inference combines these two yielding a posterior distribution that better estimates the state given the observations.

Depending on which measurements are used to form the posterior, Bayesian state estimation falls into two categories: Bayesian filtering and Bayesian smoothing. Filtering estimates the current state using all measurements up to and including the current time, making it naturally suited to online applications where estimates must be updated as new data arrive. Smoothing, by contrast, estimates the state at each time point using both past and future measurements over a fixed time window, yielding a more accurate estimate than filtering alone. This improved accuracy makes smoothing particularly valuable for offline tasks such as refining state trajectories, estimating hidden parameters, and object classification.

Computing the posterior is in general intractable. A range of algorithms has been developed for this purpose. The Kalman smoother provides an exact solution when the system dynamics and measurement model are linear and the noise is Gaussian. For nonlinear systems, the Unscented Kalman Smoother offers an approximate solution by propagating a set of deterministic sample points through the nonlinear transformations. Particle smoothers instead approximate the posterior using a set of weighted random samples, enabling the representation of highly non-Gaussian distributions, but at a computational cost that scales poorly with the state dimension. As the complexity of the state estimation problem grows, these methods become increasingly expensive, motivating the search for more computationally efficient alternatives.

The challenge is compounded by the growing number of objects present in modern airspace. In particular, unmanned aerial vehicles (UAVs) have seen increasing use

in military operations in recent years. Their small radar cross-sections and irregular flight trajectories make them difficult to detect and track reliably. As the number of objects to be monitored increases, radar systems must process a larger volume of data within a fixed computational budget, placing further demands on the efficiency of state estimation algorithms.

Machine learning has in recent years demonstrated strong capabilities in approximating complex functions from data. A model's parameters are tuned to minimise a given objective during a training phase, which may require substantial computational resources due to high-dimensional gradient computations. Once trained, however, the model consists only of efficient, simple operations, enabling fast inference at prediction time. This asymmetry between training cost and prediction cost makes machine learning models particularly attractive for problems that require repeated or real-time computation, such as state estimation in radar tracking.

Variational inference recasts posterior inference as an optimisation problem by introducing a parametrised family of distributions to approximate the intractable target distribution. Rather than computing the posterior analytically, the parameters are optimised so that the approximate distribution is as close as possible to the true smoothing distribution, typically by minimising a divergence measure between them. The parametrised distribution can, in principle, be any model that defines a sufficiently flexible family. A common choice is a Gaussian distribution, which is characterised entirely by its mean and covariance and is therefore limited in its ability to capture complex, non-Gaussian structure. In this thesis, *normalising flows* are used instead, as they parametrise an invertible and differentiable mapping from a simple base distribution to a complex target distribution, enabling the approximation of highly non-Gaussian posteriors. Specifically, a Sequential Normalising Flow (SNF) is employed to approximate the Bayesian smoothing distribution in state-space models, aiming to achieve both flexibility and computational efficiency.

1.1 Aim and objective

The aim of this thesis is to create a model that, through variational inference, can approximate the true smoothing distribution. Thereby providing a substitute for traditional methods, such as the Kalman Smoother. In addition, the model aims to achieve shorter computational times than traditional variants.

Accomplishing this consists of both developing and evaluating a model. This is achieved by introducing the Sequential Normalising Flow (SNF), a model developed in this thesis. The primary objective is to investigate the feasibility of using the SNF to approximate the Bayesian smoothing distribution. To evaluate the SNF, it is trained and evaluated for two scenarios, each defined by a distinct Gaussian state-space model. One with linear Gaussian dynamics and one with nonlinear dynamics. Training samples are generated by simulating the state-space models. The approximation quality is assessed by comparing the SNF output against refer-

ence distributions obtained from established Bayesian smoothing algorithms. The Kalman smoother for the linear Gaussian case and the Unscented Kalman smoother for the nonlinear case.

1.2 Relevance to other fields

The context of this thesis is military-inspired, however, the project is not limited to this domain. State estimation is used across a wide range of fields, and replacing traditional methods with faster, more efficient alternatives could benefit other domains. Two such examples are given below.

- **Finance:** State estimation is widely used to infer latent variables such as market trends and volatility from observed data, for example in modelling and predicting stock prices [5].
- **Robotics:** State estimation is fundamental to tasks such as simultaneous localisation and mapping (SLAM), where a robot must estimate its position and orientation from noisy sensor measurements while simultaneously building a map of its environment [15].

1.3 Thesis outline

The thesis begins in Chapter 2 with the problem setting, starting with Markov processes and state-space models. State estimation is then presented, followed by the Bayesian smoothing equations that describe the recursive solution to the state estimation problem and by established Bayesian filtering and smoothing algorithms for computing the smoothing distribution, which serve as reference methods throughout the thesis. Chapter 3 introduces variational inference, recasting the problem as an optimisation problem, with the Kullback–Leibler divergence presented as the optimisation criterion. Chapter 4 presents the machine learning models used in the thesis, beginning with neural networks, which form the foundation for all subsequent models, followed by the Gated Recurrent Unit (GRU), which enables embedding of sequential data, and normalising flows. The chapter concludes with a discussion of how these models are trained via gradient descent and backpropagation. Chapter 5 presents the methodology, detailing the two state-space models and the structure of the Sequential Normalising Flow. Chapter 6 presents the results and discusses the findings.

2

Bayesian smoothing

This chapter presents the problem formulation and theoretical framework for Bayesian state estimation. The chapter begins with Section 2.1, which presents the Markov chain and the Markov property. Section 2.2 introduces state-space models and describes the transition and measurement models. Using the Markov property, the state-space model can be expressed as a joint distribution over the states and observations. In Section 2.3, the focus shifts to inference, where we seek the distribution over states given the observations. The smoothing distribution is presented and shown to be expressed in terms of forward and backward passes. Thereafter, Section 2.4 introduces the Bayesian smoothing equations. Sections 2.5 and 2.6 introduce Bayesian filtering and smoothing algorithms. The chapter concludes by showing how the marginal likelihood can be computed using the Bayesian filtering algorithm.

2.1 Discrete-time Markov process

Assume a discrete-time stochastic process in which the state $X_t \in \mathbb{R}^d$ is indexed by the discrete time variable $t \geq 0$. The process is said to have the Markov property if the next state depends only on the current state and not on the full history of past states.

Definition 2.1 (Markov property (discrete time))**.**

A stochastic process $(X_0, X_1, X_2, \dots)$ with $X_t \in \mathbb{R}^d$ is said to have the Markov property if, for all $t \geq 1$,

$$p(x_t \mid x_{t-1}, \dots, x_0) = p(x_t \mid x_{t-1}),$$

for all $x_0, \dots, x_t \in \mathbb{R}^{d_x}$.

The Markov property is often described as memoryless, though it can equivalently be interpreted as all relevant history being encoded in the current state x_{t-1}. The conditional density $p(x_t \mid x_{t-1})$ is referred to as the transition density and is assumed to be time-homogeneous, meaning it does not depend explicitly on t. The Markov property is fundamental to this thesis, as many key definitions and derivations rely on it.

2.2 Gaussian state space model

Let $x_t \in \mathbb{R}^{d_x}$ denote a latent variable describing the state of a system at time t and let $y_t \in \mathbb{R}^{d_y}$ denote the corresponding observation. The state evolves over time according to a dynamical system governed by a transition function $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_x}$ and is observed through a measurement function $h : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$. Both the state evolution and the measurement process are corrupted by independent Gaussian noise. This setting is formalised as a State Space Model (SSM), given by

$$\begin{aligned} x_1 &\sim \mathcal{N}(\mu_0, P_0), \\ x_t &= f(x_{t-1}) + v_t, \quad v_t \sim \mathcal{N}(0, Q), \\ y_t &= h(x_t) + w_t, \quad w_t \sim \mathcal{N}(0, R), \end{aligned} \tag{2.1}$$

where v_t and w_t are mutually independent and independent across time, $Q \in \mathbb{R}^{d_x \times d_x}$ is the process noise covariance and $R \in \mathbb{R}^{d_y \times d_y}$ is the measurement noise covariance. The initial state x_1 is drawn from a Gaussian prior with mean $\mu_0 \in \mathbb{R}^{d_x}$ and covariance $P_0 \in \mathbb{R}^{d_x \times d_x}$. The additive Gaussian noise in (2.1) allows the SSM to be described equivalently in terms of probability distributions,

$$\begin{aligned} x_1 &\sim p(x_1), \\ x_t &\sim p(x_t \mid x_{t-1}), \\ y_t &\sim p(y_t \mid x_t), \end{aligned}$$

where $p(x_1) = \mathcal{N}(x_1; \mu_0, P_0)$, $p(x_t \mid x_{t-1}) = \mathcal{N}(x_t; f(x_{t-1}), Q)$, and $p(y_t \mid x_t) = \mathcal{N}(y_t; h(x_t), R)$. The transition density $p(x_t \mid x_{t-1})$ encodes the state dynamics, while the measurement density $p(y_t \mid x_t)$ describes how observations are generated from the current state. Crucially, both densities satisfy the Markov property: x_t depends only on x_{t-1}, and y_t depends only on x_t. Together, these conditional independence assumptions allow the joint distribution over all latent states and observations to be factorised as

$$p(x_{1:T}, y_{1:T}) = p(x_1) \prod_{t=2}^{T} p(x_t \mid x_{t-1}) \prod_{t=1}^{T} p(y_t \mid x_t), \tag{2.2}$$

where T denotes the sequence length and $x_{1:T} = (x_1, \ldots, x_T)$, $y_{1:T} = (y_1, \ldots, y_T)$. Given a known SSM, one can sample the joint distribution using the SSM. In practice, however, the latent states $x_{1:T}$ are unknown, and only the measurements $y_{1:T}$ are available. The problem of interest is therefore inference: estimating the latent states $x_{1:T}$ given the observations $y_{1:T}$.

2.3 State estimation via Bayesian smoothing

State estimation is the problem of estimating a latent state through the observations. Depending on the available observations, different state estimation problems arise: filtering, prediction, and smoothing. In filtering, the measurements arrive

sequentially, and the latent state is estimated using observations up to time t. Prediction shares the same setting as filtering but estimates the latent state at $t+1$ with observations up to t. In contrast, smoothing incorporates both past and future observations, thereby estimating the latent states using all available observations.

Estimating the latent state from all observations means obtaining the joint conditional distribution $p(x_{1:T} \mid y_{1:T})$. In principle, this distribution can be calculated using Bayes' rule. The distribution can be expressed as the posterior

$$p(x_{1:T} \mid y_{1:T}) = \frac{p(y_{1:T} \mid x_{1:T})p(x_{1:T})}{p(y_{1:T})}, \tag{2.3}$$

where $p(y_{1:T}) = \int p(y_{1:T} \mid x_{1:T})p(x_{1:T})\mathrm{d}x_{1:T}$ is the marginal likelihood. The prior $p(x_{1:T})$ and likelihood $p(y_{1:T} \mid x_{1:T})$ come from the SSM and factorize as

$$\begin{aligned} p(x_{1:T}) &= p(x_1)\prod_{t=2}^{T} p(x_t \mid x_{t-1}) \\ p(y_{1:T} \mid x_{1:T}) &= \prod_{t=1}^{T} p(y_t \mid x_t), \end{aligned}$$

and when multiplied together yiled (2.2). Computing (2.3) requires evaluating the marginal likelihood, an integral over all possible state trajectories. This integral is generally intractable and becomes increasingly complex as more observations are introduced. The posterior distribution can be expressed using the chain rule as

$$p(x_{1:T} \mid y_{1:T}) = p(x_1 \mid y_{1:T})\prod_{t=2}^{T} p(x_t \mid x_{1:t-1}, y_{1:T}).$$

where T is the sequence length. By the Markov property, $p(x_t \mid x_{1:t-1}) = p(x_t \mid x_{t-1})$ and $p(x_t \mid y_{1:T}) = p(x_t \mid y_{t:T})$. This results in the equation being written as

$$p(x_{1:T} \mid y_{1:T}) = p(x_1 \mid y_{1:T})\prod_{t=2}^{T} p(x_t \mid x_{t-1}, y_{t:T}).$$

where each factor depends only on the previous latent state. This yields a recursive calculation that allows the full posterior distribution to be expressed in terms of conditional distributions. The individual factors can further be decomposed into

$$p(x_t \mid x_{t-1}, y_{t:T}) \propto p(x_t \mid x_{t-1})p(y_{t:T} \mid x_t)$$

where the likelihood term can be further decomposed as

$$p(y_{t:T} \mid x_t) = p(y_t \mid x_t)p(y_{t+1:T} \mid x_t).$$

Together, these expressions reveal three distinct terms within the marginal distribution. Two of them define a forward pass and the third defines a backward pass. This shows that the smoothing distribution is obtained by using the filtering distribution as the prior:

$$p(x_t \mid x_{t-1}, y_{t:T}) \propto \underbrace{p(x_t \mid x_{t-1})\ p(y_t \mid x_t)}_{\text{forward pass}}\ \underbrace{p(y_{t+1:T} \mid x_t)}_{\text{backward pass}}.$$

It has now been shown that the Bayesian posterior distribution can be factorised via the chain rule into a product of conditional distributions, leading to a computationally efficient way to obtain the smoothing distribution. Furthermore, each factor is composed of a value from the forward pass and the backward pass. The following section presents how these passes are constructed via the Bayesian smoothing equations.

2.4 Forward-backward equations

The Bayesian smoothing equations describe how to compute the conditional distribution $p(x_t \mid x_{t-1}, y_{t:T})$ derived in the previous section. Computing this distribution requires first obtaining the filtering distribution, which is the result of a forward pass. Afterwards, a backward pass is performed, updating the filtered distribution by incorporating future measurements. The update yields the smoothing distribution, which provides a more accurate estimate than the filtering distribution alone. The following is a definition of the Bayesian smoothing equations.

Definition 2.2 (Bayesian smoothing equations)**.**

1. Starting with a prior distribution $p(x_0)$.
2. Prediction: With a dynamical model, a predictive distribution over the state x_t at time t can be obtained via the *Chapman–Kolmogorov equation*:

$$p(x_t \mid y_{1:t-1}) = \int p(x_t \mid x_{t-1}) p(x_{t-1} \mid y_{1:t-1}) \mathrm{d}x_{t-1}$$

3. Update: When a measurement y_t at time t is available, the predicted distribution can be updated. This is known as the update step. With Bayes' rule, we calculate the posterior distribution

$$p(x_t \mid y_{1:t}) = \frac{1}{Z_t} p(y_t \mid x_t) p(x_t \mid y_{1:t-1})$$

where Z_t is a normalisation factor defined as

$$Z_t = \int p(y_t \mid x_t) p(x_t \mid y_{1:t-1}) \mathrm{d}x_t$$

Store the distribution $p(x_t \mid y_{1:t})$.

4. Return to step 2 and repeat until all timesteps have been accounted for.

5. Starting at $t = T$, compute the smoothing distributions by iterating backwards. At each t, the smoothing distribution is

$$p(x_t \mid y_{1:T}) = p(x_t \mid y_{1:t}) \int \frac{p(x_{t+1} \mid x_t)\, p(x_{t+1} \mid y_{1:T})}{p(x_{t+1} \mid y_{1:t})} \, \mathrm{d}x_{t+1},$$

   which adjusts the filtering distribution $p(x_t \mid y_{1:t})$ to account for future observations.

6. Decrement t by one and return to step 5, repeating until $t = 0$.

In this definition, steps 1 to 4 constitute the Bayesian filtering equations, and the addition of steps 5 and 6 defines the Bayesian smoothing equations. Together, they are referred to as the forward filter backwards smoother, as the algorithm first computes all filtering distributions forward in time and then refines them backwards in time. The backward pass exploits the Markov property: given x_{t+1}, the future observations $y_{t+1:T}$ are conditionally independent of x_t, which makes the backward recursion possible.

Together, the Bayesian filtering and smoothing equations form a theoretical foundation for solving the state estimation problem. However, both involve integrals that are analytically tractable only under restrictive assumptions. In the following chapter, we present closed-form solutions to both the filtering and smoothing equations under the assumption of a linear Gaussian SSM, known as the Kalman filter and Kalman smoother, respectively. For nonlinear systems, where closed-form solutions are generally unavailable, the Unscented Kalman filter and Unscented Kalman smoother are presented.

2.5 Bayesian filtering algoritms

The filtering algorithms presented here are solutions for computing the distribution $p(x_t \mid y_{1:t})$. They operate recursively, with each iteration consisting of two steps: a prediction step and an update step. The choice of algorithm depends on the assumptions made about the system. For linear Gaussian systems, the Kalman filter is the optimal solution. For nonlinear systems, no closed-form optimal solution exists and we instead present the Unscented Kalman filter. The presentation of these algorithms is inspired by Simo Särkkä and Lennart Svensson book *Bayesian Filtering and Smoothing*[16].

2.5.1 Kalman filter

The Kalman filter was first presented by Rudolf Emil Kalman in 1960 in his paper *A New Approach to Linear Filtering and Prediction Problems* [11]. He derived a closed-form solution to the Bayesian filtering equations under two assumptions: that both the transition and measurement functions are linear, and that all noise is Gaussian. The filter operates on the linear Gaussian SSM,

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{v}_t, \quad \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}),$$
$$\mathbf{y}_t = \mathbf{H}\mathbf{x}_t + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}),$$

where $\mathbf{A} \in \mathbb{R}^{d_x \times d_x}$ is the transition matrix describing the linear state dynamics, and $\mathbf{H} \in \mathbb{R}^{d_y \times d_x}$ is the measurement matrix mapping the state to the observation space. Under these assumptions, both the transition density $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \mathbf{A}\mathbf{x}_{t-1}, \mathbf{Q})$ and the measurement density $p(\mathbf{y}_t \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{y}_t; \mathbf{H}\mathbf{x}_t, \mathbf{R})$ are Gaussian, which enables the filtering distribution to be computed exactly in closed form. The following algorithm describes how this is achieved.

Algorithm 2.1 (Kalman Filter)**.**

Given the initial mean $\mathbf{m}_0 \in \mathbb{R}^{d_x}$ and covariance $\mathbf{P}_0 \in \mathbb{R}^{d_x \times d_x}$ defining the prior $p(\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_0; \mathbf{m}_0, \mathbf{P}_0)$, the transition matrix $\mathbf{A} \in \mathbb{R}^{d_x \times d_x}$, the measurement matrix $\mathbf{H} \in \mathbb{R}^{d_y \times d_x}$, the process noise covariance $\mathbf{Q} \in \mathbb{R}^{d_x \times d_x}$, the measurement noise covariance $\mathbf{R} \in \mathbb{R}^{d_y \times d_y}$, and the measurement sequence $\mathbf{y}_{1:T}$, repeat the following steps for $t = 1, \ldots, T$:

1. **Predict.** Propagate the prior mean and covariance through the linear dynamics to obtain the predictive distribution $p(\mathbf{x}_t \mid \mathbf{y}_{1:t-1}) = \mathcal{N}(\mathbf{x}_t; \mathbf{m}_t^-, \mathbf{P}_t^-)$, where

$$\mathbf{m}_t^- = \mathbf{A}\mathbf{m}_{t-1},$$
$$\mathbf{P}_t^- = \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^\top + \mathbf{Q}.$$

2. **Innovate.** Compute the innovation $\mathbf{v}_t$ and its covariance $\mathbf{S}_t$,

$$\mathbf{v}_t = \mathbf{y}_t - \mathbf{H}\mathbf{m}_t^-,$$
$$\mathbf{S}_t = \mathbf{H}\mathbf{P}_t^-\mathbf{H}^\top + \mathbf{R},$$

   where $\mathbf{v}_t$ is the difference between the actual observation and the predicted observation, and $\mathbf{S}_t$ captures the total uncertainty in the innovation.

3. **Update.** Correct the predictive distribution using the innovation to obtain the filtering distribution $p(\mathbf{x}_t \mid \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{x}_t; \mathbf{m}_t^*, \mathbf{P}_t^*)$, where

$$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}^\top\mathbf{S}_t^{-1},$$
$$\mathbf{m}_t^* = \mathbf{m}_t^- + \mathbf{K}_t\mathbf{v}_t,$$
$$\mathbf{P}_t^* = \mathbf{P}_t^- - \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^\top,$$

   and $\mathbf{K}_t \in \mathbb{R}^{d_x \times d_y}$ is the Kalman gain, which weights the relative uncertainty between the predicted state and the observation.

4. **Set** $\mathbf{m}_{t-1} \leftarrow \mathbf{m}_t^*$ and $\mathbf{P}_{t-1} \leftarrow \mathbf{P}_t^*$ and proceed to the next time step.

For linear Gaussian systems, the Kalman filter computes the exact filtering distribution in closed form and is therefore the Minimum Mean-Square Error (MMSE) estimator. When nonlinear dynamics are introduced, however, the exact filtering distribution is no longer tractable, and the Kalman filter can no longer achieve this optimal solution. Approximate methods must therefore be employed, one of which is the Unscented Kalman Filter, presented in the following section.

2.5.2 Unscented Kalman filter

The Unscented Kalman Filter (UKF) is designed to handle nonlinear state space models of the form

$$\begin{aligned}\mathbf{x}_t &= f(\mathbf{x}_{t-1}) + \mathbf{v}_t, \quad \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}),\\ \mathbf{y}_t &= h(\mathbf{x}_t) + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}),\end{aligned}$$

where $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_x}$ is the nonlinear transition function and $h : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ is the nonlinear measurement function. As in the linear case, $\mathbf{v}_t$ and $\mathbf{w}_t$ are mutually independent Gaussian noise terms, independent across time.

The UKF's ability to handle non-linear dynamics stems from the Unscented Transform (UT). Rather than linearising the nonlinear functions directly, the UT represents the prior distribution through a set of deterministically chosen points, called sigma points, that capture its mean and covariance. These sigma points are propagated through the non-linear functions, yielding an approximation of the predicted distribution. The sigma points are positioned as

$$\begin{aligned}\mathcal{X}_{t-1}^{(0)} &= \mathbf{m}_{t-1},\\ \mathcal{X}_{t-1}^{(i)} &= \mathbf{m}_{t-1} + \sqrt{n+\lambda}\left[\mathbf{P}_{t-1}^{1/2}\right]_i,\\ \mathcal{X}_{t-1}^{(i+n)} &= \mathbf{m}_{t-1} - \sqrt{n+\lambda}\left[\mathbf{P}_{t-1}^{1/2}\right]_i,\end{aligned}$$

for $i = 1, \ldots, n$, where $\mathbf{m}_{t-1}$ is the mean and $\mathbf{P}_{t-1}$ is the covariance of the previous filtering distribution. The matrix $\mathbf{P}_{t-1}^{1/2}$ denotes the Cholesky factor of $\mathbf{P}_{t-1}$, satisfying $\mathbf{P}_{t-1} = \mathbf{P}_{t-1}^{1/2}(\mathbf{P}_{t-1}^{1/2})^\top$, and $[\cdot]_i$ denotes its i-th column. For a state of dimension n, this yields $2n+1$ sigma points in total. The scaling parameter λ controls the spread of the sigma points around the mean and is given by

$$\lambda = \alpha^2(n+\kappa) - n,$$

where α and κ are tuning parameters. Each sigma point is assigned a mean weight $W_i^{(m)}$ and a covariance weight $W_i^{(c)}$, given by

$$\begin{aligned}W_0^{(m)} &= \frac{\lambda}{n+\lambda},\\ W_0^{(c)} &= \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta),\\ W_i^{(m)} &= W_i^{(c)} = \frac{1}{2(n+\lambda)}, \quad i = 1, \ldots, 2n,\end{aligned} \tag{2.4}$$

where β is a parameter that incorporates prior knowledge of the distribution. The weights within each class sum to unity. Compared to a Monte Carlo approach, in which a large number of randomly sampled particles must be propagated to achieve a reliable estimate, the UT achieves an accurate approximation of the mean and covariance using only $2n + 1$ deterministic sigma points. With the UT defined, the UKF algorithm is presented below.

Algorithm 2.2 (Unscented Kalman Filter)**.**

Given the initial mean $\mathbf{m}_0 \in \mathbb{R}^{d_x}$ and covariance $\mathbf{P}_0 \in \mathbb{R}^{d_x \times d_x}$ defining the prior $p(\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_0; \mathbf{m}_0, \mathbf{P}_0)$, the transition function f, the measurement function h, the process noise covariance $\mathbf{Q} \in \mathbb{R}^{d_x \times d_x}$, the measurement noise covariance $\mathbf{R} \in \mathbb{R}^{d_y \times d_y}$, and the measurement sequence $\mathbf{y}_{1:T}$, repeat the following steps for $t = 1, \ldots, T$:

1. **Sigma points.** Compute the $2n+1$ sigma points from the previous filtering distribution with mean $\mathbf{m}_{t-1}$ and covariance $\mathbf{P}_{t-1}$ according to (2.4).
2. **Predict.** Propagate the sigma points through the non-linear transition function,

$$\mathcal{X}_t^{(i)} = f\left(\mathcal{X}_{t-1}^{(i)}\right), \quad i = 0, \ldots, 2n,$$

   and compute the predictive mean and covariance,

$$\begin{aligned}
\mathbf{m}_t^- &= \sum_{i=0}^{2n} W_i^{(m)}\, \mathcal{X}_t^{(i)}, \\
\mathbf{P}_t^- &= \sum_{i=0}^{2n} W_i^{(c)} \left(\mathcal{X}_t^{(i)} - \mathbf{m}_t^-\right)\left(\mathcal{X}_t^{(i)} - \mathbf{m}_t^-\right)^\top + \mathbf{Q},
\end{aligned}$$

   defining the predictive distribution $p(\mathbf{x}_t \mid \mathbf{y}_{1:t-1}) \approx \mathcal{N}(\mathbf{x}_t;\, \mathbf{m}_t^-, \mathbf{P}_t^-)$.
3. **Innovate.** Propagate the sigma points through the nonlinear measurement function,

$$\mathcal{Y}_t^{(i)} = h\left(\mathcal{X}_t^{(i)}\right), \quad i = 0, \ldots, 2n,$$

   and compute the predicted observation mean $\boldsymbol{\mu}_t$, the innovation covariance $\mathbf{S}_t$, and the cross-covariance $\mathbf{C}_t$,

$$\begin{aligned}
\boldsymbol{\mu}_t &= \sum_{i=0}^{2n} W_i^{(m)}\, \mathcal{Y}_t^{(i)}, \\
\mathbf{S}_t &= \sum_{i=0}^{2n} W_i^{(c)} \left(\mathcal{Y}_t^{(i)} - \boldsymbol{\mu}_t\right)\left(\mathcal{Y}_t^{(i)} - \boldsymbol{\mu}_t\right)^\top + \mathbf{R}, \\
\mathbf{C}_t &= \sum_{i=0}^{2n} W_i^{(c)} \left(\mathcal{X}_t^{(i)} - \mathbf{m}_t^-\right)\left(\mathcal{Y}_t^{(i)} - \boldsymbol{\mu}_t\right)^\top.
\end{aligned}$$

4. **Update.** Correct the predictive distribution using the observation $\mathbf{y}_t$ to obtain the filtering distribution $p(\mathbf{x}_t \mid \mathbf{y}_{1:t}) \approx \mathcal{N}(\mathbf{x}_t; \mathbf{m}_t^*, \mathbf{P}_t^*)$, where

$$\begin{aligned}\mathbf{K}_t &= \mathbf{C}_t \mathbf{S}_t^{-1}, \\ \mathbf{m}_t^* &= \mathbf{m}_t^- + \mathbf{K}_t \left(\mathbf{y}_t - \boldsymbol{\mu}_t\right), \\ \mathbf{P}_t^* &= \mathbf{P}_t^- - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top,\end{aligned}$$

   and $\mathbf{K}_t \in \mathbb{R}^{d_x \times d_y}$ is the Kalman gain.
5. **Set** $\mathbf{m}_{t-1} \leftarrow \mathbf{m}_t^*$ and $\mathbf{P}_{t-1} \leftarrow \mathbf{P}_t^*$ and proceed to the next time step.

The UKF is well-suited to non-linear systems and can approximate the filtering distribution using the UT. For non-linear systems, other solutions include the Extended Kalman filter (EKF). The EKF operates by linearization and therefore requires the derivatives of the transiton function and measurement functions. This is something the UKF avoids by using the UT. The UKF, compared to the EKF, is computationally more expensive because each step requires Cholesky factorisation.

2.6 Bayesian smoothing algoritms

Thus far, the focus has been on computing the filtering distribution $p(x_t \mid y_{1:t})$. When the full observation sequence $y_{1:T}$ is available, we can instead seek the smoothing distribution $p(x_t \mid y_{1:T})$, which incorporates both past and future observations, obtaining a more accurate estimate at each time step. The smoothing algorithms presented here build directly on the filtering algorithms of the previous section. The filter is first run forward in time to obtain the filtering distribution $p(x_t \mid y_{1:t})$, after a backward pass is performed to refine these estimates by incorporating future observations, yielding the smoothing distribution $p(x_t \mid y_{1:T})$.

2.6.1 Unscented Kalman smoother

The Unscented Kalman Smoother (UKS) is an approximated solution to the Bayesian smoothing equations. The UKS first requires calculating the filtered distribution using the UKF, as presented in the section above. With these distributions, the backward pass can be performed. The recursion is initialised at time T using the final filtering distribution and proceeds backwards to $t = 1$. The following is a definition for the Unscented Kalman smoother.

Algorithm 2.3 (Unscented Kalman Smoother)**.**

Given the filtering means $\mathbf{m}_t$ and covariances $\mathbf{P}_t$ for $t = 0, \ldots, T$, obtained from the Unscented Kalman Filter (Algorithm 2.2), and the process noise covariance $\mathbf{Q} \in \mathbb{R}^{d_x \times d_x}$, initialise the smoothed mean and covariance at the final time step as $\mathbf{m}_T^s = \mathbf{m}_T$ and $\mathbf{P}_T^s = \mathbf{P}_T$, and repeat the following steps for $t = T - 1, \ldots, 0$:

1. **Sigma points.** Compute the $2n+1$ sigma points from the filtering distribution with mean $\mathbf{m}_t$ and covariance $\mathbf{P}_t$ according to (2.4).
2. **Predict.** Propagate the sigma points through the nonlinear transition function,

$$\mathcal{X}_{t+1}^{(i)} = f\left(\mathcal{X}_t^{(i)}\right), \quad i = 0, \ldots, 2n,$$

and compute the predictive mean, covariance, and cross-covariance,

$$\begin{aligned}
\mathbf{m}_{t+1}^- &= \sum_{i=0}^{2n} W_i^{(m)}\, \mathcal{X}_{t+1}^{(i)}, \\
\mathbf{P}_{t+1}^- &= \sum_{i=0}^{2n} W_i^{(c)} \left(\mathcal{X}_{t+1}^{(i)} - \mathbf{m}_{t+1}^-\right)\left(\mathcal{X}_{t+1}^{(i)} - \mathbf{m}_{t+1}^-\right)^\top + \mathbf{Q}, \\
\mathbf{D}_t &= \sum_{i=0}^{2n} W_i^{(c)} \left(\mathcal{X}_t^{(i)} - \mathbf{m}_t\right)\left(\mathcal{X}_{t+1}^{(i)} - \mathbf{m}_{t+1}^-\right)^\top,
\end{aligned}$$

defining the predictive distribution $p(\mathbf{x}_{t+1} \mid \mathbf{y}_{1:t}) \approx \mathcal{N}(\mathbf{x}_{t+1};\, \mathbf{m}_{t+1}^-, \mathbf{P}_{t+1}^-)$.
3. **Smooth.** Correct the filtering distribution using the smoothed estimate at $t+1$ to obtain the smoothing distribution $p(\mathbf{x}_t \mid \mathbf{y}_{1:T}) \approx \mathcal{N}(\mathbf{x}_t;\, \mathbf{m}_t^s, \mathbf{P}_t^s)$, where

$$\begin{aligned}
\mathbf{G}_t &= \mathbf{D}_t \left[\mathbf{P}_{t+1}^-\right]^{-1}, \\
\mathbf{m}_t^s &= \mathbf{m}_t + \mathbf{G}_t \left(\mathbf{m}_{t+1}^s - \mathbf{m}_{t+1}^-\right), \\
\mathbf{P}_t^s &= \mathbf{P}_t + \mathbf{G}_t \left(\mathbf{P}_{t+1}^s - \mathbf{P}_{t+1}^-\right) \mathbf{G}_t^\top,
\end{aligned}$$

and $\mathbf{G}_t \in \mathbb{R}^{d_x \times d_x}$ is the smoothing gain. Compared to the Kalman gain $\mathbf{K}_t$ in the filtering step, which balances the influence of the observation and the prior, the smoothing gain $\mathbf{G}_t$ balances the influence of the filtering distribution at time t and the predictive distribution at time $t+1$.
4. **Set** $\mathbf{m}_{t+1}^s \leftarrow \mathbf{m}_t^s$ and $\mathbf{P}_{t+1}^s \leftarrow \mathbf{P}_t^s$ and proceed to the previous time step.

The UKS thus provides a tractable approximation to the smoothing distribution for non-linear systems, extending the UKF with a single backward pass at a small additional cost.

2.7 Marginal likelihood

The marginal likelihood $p(y_{1:T})$ appears as the normalising constant in Bayes ' theorem when computing the full smoothing distribution $p(x_{1:T} \mid y_{1_T})$. As presented above, this constant is generally intractable as it requires integrating over all possible states

$$p(y_{1:T}) = \int p(x_{1:T}, y_{1:T})\, \mathrm{d}x_{1:T}.$$

However, using the UKF, an approximation of the marginalised likelihood can be obtained. Via the chain rule, the marginal likelihood can be expressed by the factorisation as

$$p(y_{1:T}) = \prod p(y_t \mid y_{1:t-1}).$$

where each factor $p(y_t \mid y_{1:t-1})$ is the predictive likelihood of observation y_t given all past observations. This can be obtained by marginalising the predictive state distribution

$$p(y_t \mid y_{1:t-1}) = \int p(y_t \mid x_t) p(x_t \mid y_{1:t-1}) \, \mathrm{d}x_t.$$

where both terms can be obtained from the UKF. Since the UKF approximates distributions with Gaussians, the integral reduces to a Gaussian. Therefore, the solution becomes closed-form. The predictive state distribution is approximated as

$$p(x_t \mid y_{1:t-1}) \approx \mathcal{N}(x_t \mid x_{t|t-1}, P_{t|t-1})$$

and the predictive observation mean μ_t and innovation covariance S_t are computed from the forward propagated sigma points as

$$\begin{aligned} \mu_t &= h(x_{t|t-1}) \\ S_t &= \sum W_i (\mathcal{Y}_k^i - \mu_k)(\mathcal{Y}_k^i - \mu_k)^T + R_{k-1} \end{aligned}$$

giving the predictive likelihood

$$p(y_t \mid y_{1:t-1}) = \mathrm{N}(y_t \mid y_{t|t-1}, S_t).$$

The approximated marginal likelihood is then computed as the product of the predictive likelihood over all timesteps as

$$p(y_{1:T}) = \prod_{i=1}^{T} p(y_t \mid y_{t|t-1}, S_t).$$

Since the marginal likelihood is a product of many probability densities, it is numerically advantageous to work on the logarithmic scale, converting the product into a sum to avoid potential numerical instability. It should be noted that for nonlinear systems, the marginal likelihood obtained through the UKF is an approximation, a consequence of the Gaussian approximation made at each step of the filter. For linear Gaussian systems, the marginal likelihood can be computed exactly by replacing the UKF with the Kalman filter, following the same procedure outlined above.

3
Variational inference

This chapter presents variational inference as an alternative approach to obtaining the intractable smoothing distribution. Rather than computing the distribution directly, a parametrised variational distribution is introduced and optimised to approximate the target, recasting the intractable inference problem as an optimisation problem.

The chapter begins by introducing variational inference and the Kullback–Leibler (KL) divergence as the optimisation objective. The asymmetry of the KL divergence is then examined through the forward and reverse KL divergences, which differ in how they penalise discrepancies between the variational and target distributions, leading to qualitatively different approximation behaviours. Thereafter, the evidence lower bound (ELBO) is derived, showing how the intractable posterior can be bypassed to yield a tractable optimisation objective.

3.1 Variational inference

Variational inference is a method for approximating probability distributions by recasting inference as an optimisation problem [10]. The core idea is to approximate a target distribution p with a parametrised variational distribution q_ϕ, where ϕ denotes the variational parameters. The optimal parameters are found by minimising a measure of discrepancy between q_ϕ and p,

$$\phi^* = \arg\min_\phi \mathcal{D}_{\mathrm{KL}}(q_\phi \,\|\, p)\,.$$

A common measure of discrepancy between two distributions is the *Kullback–Leibler* (KL) divergence. Given two continuous distributions q_ϕ and p with densities defined on the same domain, and assuming absolute continuity $q_\phi \ll p$ (ensuring $p(x) = 0$ implies $q_\phi(x) = 0$, so that the ratio $q_\phi(x)/p(x)$ is well defined), the KL divergence is given by

$$\mathcal{D}_{\mathrm{KL}}(q_\phi \,\|\, p) = \int q_\phi(x) \log\left(\frac{q_\phi(x)}{p(x)}\right) \mathrm{d}x = \mathbb{E}_{x\sim q_\phi}\left[\log q_\phi(x) - \log p(x)\right].$$

The KL divergence has two important properties. First, it is non-negative and equals zero if and only if $q_\phi = p$ everywhere, meaning the optimisation objective reaches its minimum precisely when the approximation is exact. Secondly, the KL divergence is asymmetric, meaning that $\mathcal{D}_{\mathrm{KL}}(q_\phi \,\|\, p) \neq \mathcal{D}_{\mathrm{KL}}(p \,\|\, q_\phi)$ in general. This

asymmetry gives rise to two distinct variational objectives, the forward and reverse KL divergences, whose differing properties and implications for the approximation are discussed in the following section.

3.2 Forward and reverse Kullback–Leibler

In practice, distributions are often not available in closed form, and only sampling and density evaluation are possible. The asymmetry of the KL divergence means that the choice of which distribution to sample from gives rise to two distinct variants of the optimisation objective. Minimising $\mathcal{D}_{\mathrm{KL}}(p \,\|\, q_\phi)$, known as the *forward KL*,

$$\phi^* = \arg\min_\phi \mathcal{D}_{\mathrm{KL}}(p \,\|\, q_\phi) = \arg\min_\phi \mathbb{E}_{x\sim p}\left[\log p(x) - \log q_\phi(x)\right],$$

requires samples from the target p. Minimising $\mathcal{D}_{\mathrm{KL}}(q_\phi \,\|\, p)$, known as the *reverse KL*,

$$\phi^* = \arg\min_\phi \mathcal{D}_{\mathrm{KL}}(q_\phi \,\|\, p) = \arg\min_\phi \mathbb{E}_{x\sim q_\phi}\left[\log q_\phi(x) - \log p(x)\right],$$

instead requires samples from the variational distribution q_ϕ, which is accessible by construction. The two variants differ fundamentally in how the variational distribution distributes its probability mass. The forward KL is *mean-seeking*, since it penalises regions where $p(x) > 0$ but $q_\phi(x) \approx 0$, the approximation is encouraged to spread mass across all modes of p. The reverse KL is *mode-seeking* and instead penalises regions where $q_\phi(x) > 0$ but $p(x) \approx 0$, the approximation is encouraged to concentrate mass on a single high-probability region of p, avoiding placing mass where p is low.

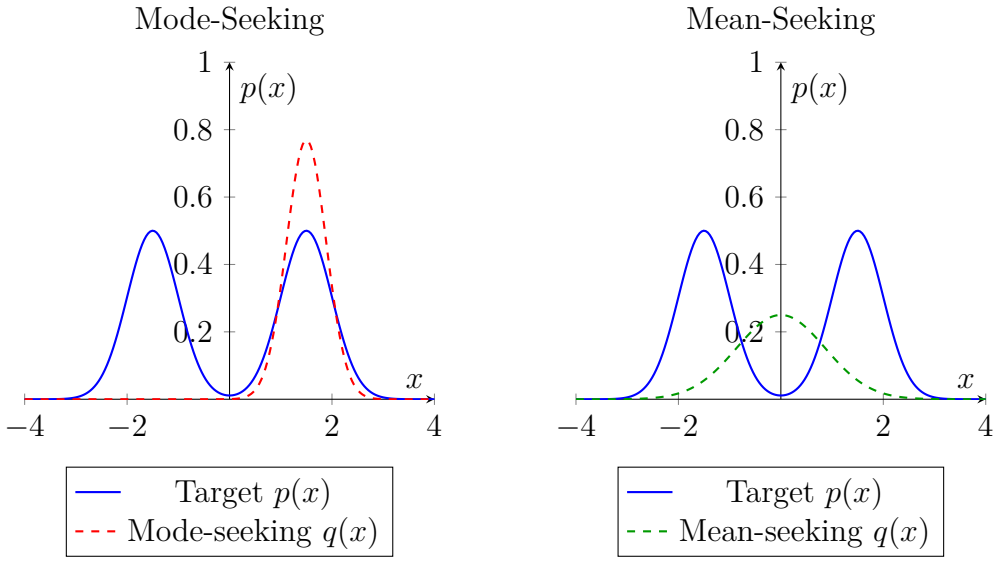


Figure 3.1: Illustration of mode-seeking vs. mean-seeking behavior for a bimodal target distribution.

This distinction becomes particularly pronounced when the target distribution is multimodal. Figure 3.1 illustrates both variants when approximating a bimodal

distribution with a parametric Gaussian. In the mode-seeking case, the Gaussian concentrates around one mode, as placing mass in the low-density region between the modes would be heavily penalised by the reverse KL. In the mean-seeking case, the Gaussian spreads across both modes, covering the full support of p at the cost of also assigning mass to the low-density region between them.

3.3 Evidence lower bound

Minimising the KL divergence requires evaluating the intractable smoothing distribution $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})$. By defining $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T}) = p(\mathbf{x}_{1:T}, \mathbf{y}_{1:T})/p(\mathbf{y}_{1:T})$, allows the KL divergence to be expanded as

$$\begin{aligned}
&\mathcal{D}_{\mathrm{KL}}(q_\phi(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T}) \,\|\, p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})) \\
&\quad = \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T}) - \log p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})] \\
&\quad = \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T}) - \log p(\mathbf{x}_{1:T}, \mathbf{y}_{1:T}) + \log p(\mathbf{y}_{1:T})] \\
&\quad = \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T}) - \log p(\mathbf{x}_{1:T}, \mathbf{y}_{1:T})] + \log p(\mathbf{y}_{1:T}),
\end{aligned} \tag{3.1}$$

where $\log p(\mathbf{y}_{1:T})$ is pulled outside the expectation since it does not depend on $\mathbf{x}_{1:T}$. Although $\log p(\mathbf{y}_{1:T})$ is constant with respect to ϕ, it is generally intractable and cannot be evaluated directly, meaning the KL divergence cannot be minimised directly. Rearranging (3.1) yields

$$\begin{aligned}
\log p(\mathbf{y}_{1:T}) = \mathcal{D}_{\mathrm{KL}}(q_\phi(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T}) \,\|\, p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})) \\
+ \mathbb{E}_{q_\phi}[\log p(\mathbf{x}_{1:T}, \mathbf{y}_{1:T}) - \log q_\phi(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})]\,.
\end{aligned} \tag{3.2}$$

Since $\log p(\mathbf{y}_{1:T})$ is constant with respect to ϕ and $\mathcal{D}_{\mathrm{KL}} \geq 0$, the second term on the right-hand side constitutes a lower bound on $\log p(\mathbf{y}_{1:T})$. This term is known as the *Evidence Lower Bound* (ELBO),

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi}[\log p(\mathbf{x}_{1:T}, \mathbf{y}_{1:T}) - \log q_\phi(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})] \leq \log p(\mathbf{y}_{1:T}).$$

Maximising $\mathcal{L}(\phi)$ is equivalent to minimising the KL divergence, since the two sum to the constant $\log p(\mathbf{y}_{1:T})$ by (3.2). This is the key insight of variational inference: rather than minimising the KL divergence directly, which requires evaluating the intractable posterior $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})$, one can instead maximise the ELBO, which depends only on the joint distribution $p(\mathbf{x}_{1:T}, \mathbf{y}_{1:T})$ and the variational distribution q_ϕ, both of which are tractable.

4

Machine learning models

This chapter introduces different machine learning models. These serve as building blocks for the model that is developed in this thesis. Machine learning models are parametrised functions whose parameters are optimised to approximate a target function or a probability distribution. Building on the variational inference framework introduced in the previous chapter, we now present the specific model architectures that realise this approximation in practice.

The chapter begins with neural networks and their fundamental unit, the perceptron, which underlies all models presented here. Then, recurrent neural networks are introduced, extending basic neural networks to handle sequential data by capturing temporal dependencies, with a particular focus on the Gated Recurrent Unit (GRU). Next, normalising flows are presented, which create an invertible mapping from a simple base distribution to a more complex target distribution. The chapter concludes by discussing how these models are trained via gradient descent and how backpropagation updates their parameters.

4.1 Neural networks

Neural networks are parametrised models that learn to approximate a mapping $f : \mathbb{R}^n \to \mathbb{R}^m$, where n and m are integers. The number of parameters is determined by the chosen architecture, and with a sufficiently large number of parameters, the model can approximate difficult functions. A neural network is composed of several interconnected units called perceptrons, which are the fundamental building blocks of all parametrised models of this type. This section begins by presenting the perceptron, before discussing how multiple perceptrons can be combined to construct a *Fully Connected Neural Network* (FCNN).

4.1.1 Perceptron

An individual perceptron takes an input vector $\mathbf{x} = \mathbb{R}^n$ and produces a scalar output y, expressed as

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where the input undergoes a linear transformation via the weights w_i and bias b, hence shifting and scaling the input. Since the ability to approximate a non-linear

function is favourable, the linear transformation is followed by a non-linear activation function, denoted as σ. Training the perceptron is achieved by defining w and b as parameters. Figure 4.1 illustrates this computation graphically.

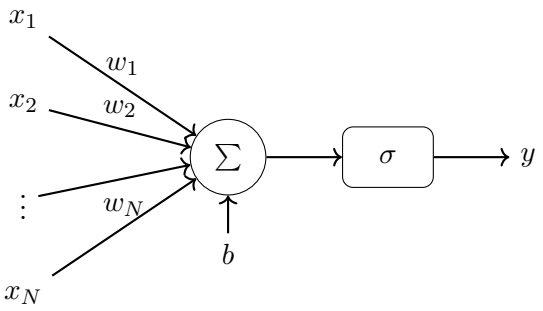


Figure 4.1: Illustration of a perceptron with weighted inputs, bias, summation, and activation function.

A single perceptron is limited in its expressive capability, as it produces a scalar output and can only represent a single non-linear function of its inputs. To increase expressiveness, multiple perceptrons are grouped into layers, and multiple layers are stacked in sequence. This construction yields the *Fully Connected Neural Network* (FCNN), which is presented next.

4.1.2 Fully connected neural network

A *Fully Connected Neural Network* (FCNN) is a structure consisting of multiple perceptrons arranged in layers. The number of perceptrons per layer is referred to as the width, and the number of layers is referred to as the depth. A single perceptron defines a mapping $\mathrm{P} : \mathbb{R}^m \to \mathbb{R}^1$, the FCNN defines a more general mapping $\mathrm{NN} : \mathbb{R}^m \to \mathbb{R}^n$. The first and last layers are called the input and output layers, respectively, and the layers in between are called hidden layers. The input is propagated sequentially through all layers, so that each layer receives the output of the previous one. An illustration of an FCNN with three inputs, three hidden layers, and two outputs is shown in Figure 4.2.

Compared to a single perceptron, the FCNN has considerably greater expressive power. The activation function σ introduces nonlinearity into the network, which is essential. Without it, each layer would perform only a linear transformation, and the entire network could be collapsed into a single equivalent linear mapping, rendering depth meaningless. Due to this nonlinearity, an FCNN can in principle approximate any continuous function, as stated by the *Universal Approximation Theorem*. In 1989, George Cybenko showed in *Approximation by Superpositions of a Sigmoidal Function* that a single hidden layer with a sigmoid activation function suffices to approximate any continuous function on a compact domain [2]. Later that same year, *Multilayer Feedforward Networks are Universal Approximators* extended this result, showing that the approximation property is not specific to the sigmoid but

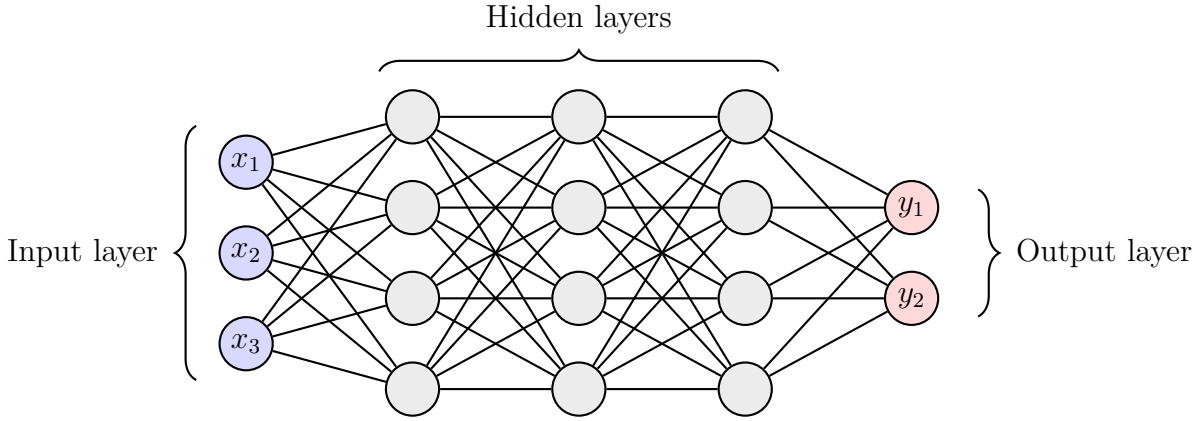


Figure 4.2: Fully connected feedforward neural network with input layer, three hidden layers, and output layer.

holds for any non-linear activation function [9]. An entire proof is beyond the scope of this thesis, but a simplified statement is provided below.

Theorem 4.1 (Universal approximation theorem)**.**
Let $I \subset \mathbb{R}^n$ be a compact set and $\sigma : \mathbb{R} \to \mathbb{R}$ a non-constant, non-linear, bounded, and continuous function. Then the set of functions $G : I \to \mathbb{R}$ defined by

$$G(x) = \sum_{i=1}^{N} \alpha_i \, \sigma(w_i^T x + \theta_i) \quad (x \in I)$$

and the constants $w_i \in \mathbb{R}^n$, $\theta_i, \alpha_i \in \mathbb{R}$, is dense in $C(I)$. That is, for each continuous function $f : I$ and $\epsilon > 0$, there exists a G of the above form such that

$$\sup_{x \in I} |G(x) - f(x)| < \epsilon.$$

While the universal approximation theorem guarantees that an FCNN can approximate any continuous function, it does not specify the size of the network required or the amount of training data and computation needed to achieve this. It therefore does not account for practical constraints such as model complexity, computational cost, or sample size.

So far, the activation function has been discussed only in terms of the property it must satisfy, namely, nonlinearity. We now present a selection of activation functions that are particularly relevant to the models used in this thesis.

4.1.3 Activation function

Commonly used activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). In this thesis, the tanh and LeakyReLU activation functions are used. The tanh is defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

and maps any real input to the range $(-1, 1)$. The LeakyReLU is defined as

$$\text{LeakyReLU}(x) = \begin{cases} kx & \text{if } x < 0 \\ x & \text{if } x \geq 0, \end{cases}$$

where k determines the slope, which, when equal to 0, becomes the commonly used ReLU function. The LeakyReLU is a variant of the standard ReLU that permits a small, non-zero gradient for negative inputs. Thereby addressing a known limitation of the standard ReLU called the dying ReLU problem, in which neurons become permanently inactive during training as the gradient equals zero for all negative inputs.

4.2 Recurrenct neural network

So far, neural networks have been presented as mappings of the form $NN : \mathbb{R}^n \to \mathbb{R}^m$. Now consider a sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$, where $\mathbf{x}_t \in \mathbb{R}^n$, whose elements exhibit temporal dependencies. Although an FCNN could process each $\mathbf{x}_t$ individually, it lacks a mechanism to retain information across time steps, so the temporal structure of the sequence would be lost. We therefore require an architecture capable of mapping the full sequence to an output, $\mathbb{R}^{n \times T} \to \mathbb{R}^m$.

Several architectures are capable of processing sequential data. This thesis focuses on *Recurrent Neural Networks* (RNNs), a class of models designed to handle sequential inputs by maintaining an internal hidden state that is updated as new data arrives. The most widely used RNN architectures are the Long Short-Term Memory (LSTM) network and the *Gated Recurrent Unit* (GRU), both of which employ gating mechanisms to control the flow of information through the hidden state. This allows the model to selectively retain or discard information from previous time steps, producing an embedding that captures temporal dependencies across the sequence. The GRU is generally considered a simplified variant of the LSTM with comparable performance [1], and is therefore the architecture adopted in this thesis.

4.2.1 Gated recurrent unit

The GRU processes a sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$, consisting of vectors $\mathbf{x}_t \in \mathbb{R}^d$ at time t. The model works through the sequence sequentially. The key component is to retain information about the previous data point when processing the next, hence accounting for temporal dependencies. At each time step, the GRU outputs the embedding $h_t \in \mathbb{R}^e$. During the recurrent processing, the same architecture is used at each time step. Figure 4.3 illustrates the recurrent computation and the computation that is described below. The GRU consists of two gates. One is called the *Reset gate* and the other *Update gate*. The reset gate controls the extent to which the previous hidden state is forgotten when computing the new candidate

state. It takes as input $\mathbf{x}_t$ and an embedded representation $\mathbf{h}_{t-1}$ of $\mathbf{x}_{t-1}$. For the first data point $\mathbf{h}_{t-1}$ is commonly set to a vector filled with zeros. The reset gate is defined as

$$r_t = \sigma(\mathbf{W}_{ir}\mathbf{x}_t + \mathbf{b}_{ir} + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_{hr})$$

where $\mathbf{W}_{ir} \in \mathbb{R}^{e\times d}$ and $\mathbf{W}_{hr} \in \mathbb{R}^{e\times e}$ are weights matrices and $\mathbf{b}_{ir} \in \mathbb{R}^e$ and $\mathbf{b}_{hr} \in \mathbb{R}^e$ are biases. The output $\mathbf{r_t}$ represents how much of $\mathbf{h}_{t-1}$ is used. After the *Reset Gate*, the *Candidate hidden state* is calculated. This is an initial representation of the embedding output. Using $\mathbf{r}_t$, $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ as inputs the function is defined as

$$\mathbf{h}_t^* = \tanh(\mathbf{W}_{in}\mathbf{x}_t + \mathbf{b}_{in} + \mathbf{r}_t \odot (\mathbf{W}_{hn}\mathbf{h}_{t-1} + \mathbf{b}_{hn}))$$

where $\odot$ is the element-wise product and $\mathbf{W}_{in} \in \mathbb{R}^{e\times d}$ and $\mathbf{W}_{hn} \in \mathbb{R}^{e\times e}$ are weights matrices and $\mathbf{b}_{in} \in \mathbb{R}^e$ and $\mathbf{b}_{hn} \in \mathbb{R}^e$ are biases. The output h_t^* is the *Candidate hidden state*. The next step is the *Update gate*, which decides the balance between the *Candidate hidden state* and the previous hidden state. While the update gate is similar to the reset gate, the weights are different and is defined as

$$\mathbf{z}_t = \sigma(\mathbf{W}_{iz}\mathbf{x}_t + \mathbf{b}_{iz} + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_{hz}).$$

where $\mathbf{W}_{iz} \in \mathbb{R}^{e\times d}$ and $\mathbf{W}_{hz} \in \mathbb{R}^{e\times e}$ are weights matrices and $\mathbf{b}_{iz} \in \mathbb{R}^e$ and $\mathbf{b}_{hz} \in \mathbb{R}^e$ are biases. The last step is calculating the new hidden state. With the output from the update gate, the new hidden state is calculated as

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_t^* + \mathbf{z}_t \odot \mathbf{h}_{t-1}$$

where $\mathbf{h}_{t-1}$ is the previus hidden state and $\mathbf{h}_t^*$ is the candinate hidden state. The equations presented above define a single GRU unit and computes the hidden state for a single step. By stacking units, the hidden state is updated with new information, yielding a representation of all previous states.

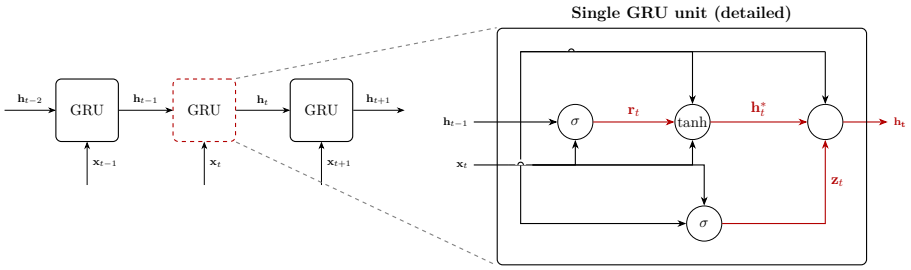


Figure 4.3: GRU architecture. Left: Recurrent computation. Right: Computation for each recurrent.

4.3 Constructuing ditributions

Using neural network-based models to approximate or construct distributions has a wide range of applications. Several model families exist for this purpose. Variational Autoencoders (VAEs) employ an encoder-decoder architecture to map data to

and from a latent space distribution [12]. Generative Adversarial Networks (GANs) operate through an adversarial process between a generator and a discriminator, producing samples that resemble the training data [7]. Diffusion models, popular for image generation, can learn a transformation that maps images to noise. Then, by reversing the transformation, we instead gain a model capable of generating images from only noise [8]. While these models implicitly construct a distribution over the data, they cannot explicitly evaluate the likelihood of a given sample. A model which can evaluate likelihoods is a normalising flow.

4.3.1 Normalising flow

Let p_Z and p_X be probability distributions living on spaces $\mathcal{Z}$ and $\mathcal{X}$, respectively. A normalising flow is a differentiable, parametrised, bijective function $M : \mathbb{R}^d \to \mathbb{R}^d$ that maps samples z from p_Z to samples x from p_X. Since M can be chosen flexibly, a simple distribution p_Z can be transformed into a rich distribution p_X. Because M is bijective, the inverse mapping $z = M^{-1}(x)$ also exists, so samples can equally be mapped back from $\mathcal{X}$ to $\mathcal{Z}$. The base distribution p_Z is chosen so that both sampling and density evaluation are straightforward, whereas the target distribution p_X is typically intractable. It can be evaluated but not sampled from directly. By parameterisation, the flow can be shaped so that samples produced through M follow the target distribution p_X. This transformation is written as

$$x = M(z), \quad z \sim p_Z. \tag{4.1}$$

Equation (4.1) only describes how individual points are mapped from the base space to the target space; it does not by itself give the density $p_X(x)$. To obtain $p_X(x)$, the base density $p_Z(z)$ must be transformed to account for the local change of volume induced by M, subject to the constraint that total probability mass is conserved: since p_Z and p_X are both probability measures, $\int p_Z(z)\,dz = 1$ and $\int p_X(x)\,dx = 1$, and the transformation can redistribute probability mass but never change its total. Since M is differentiable, the change-of-variables formula gives the density at a point $x = M(z)$ as

$$p_X(x) = p_Z(z)\left|\det J_M(z)\right|^{-1}, \tag{4.2}$$

where $J_M(z)$ is the Jacobian of M at z, i.e. the matrix of partial derivatives of each output dimension with respect to each input dimension,

$$J_M(z) = \begin{bmatrix} \dfrac{\partial M_1}{\partial z_1} & \cdots & \dfrac{\partial M_1}{\partial z_d} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial M_d}{\partial z_1} & \cdots & \dfrac{\partial M_d}{\partial z_d} \end{bmatrix}.$$

Intuitively, the Jacobian determinant captures the local change in volume induced by the transformation M at a given point. As M reshapes the space, the determinant of the Jacobian $\det J_M(z)$ quantifies how volumes are stretched or compressed at z, ensuring that the total probability mass is conserved under the transformation.

An illustration of how M bends the space into the target distribution is shown in Figure 4.4. The base distribution is bent to accommodate the banana distribution. The grid shows how the probability mass is stretched and squeezed to best approximate the target distribution.

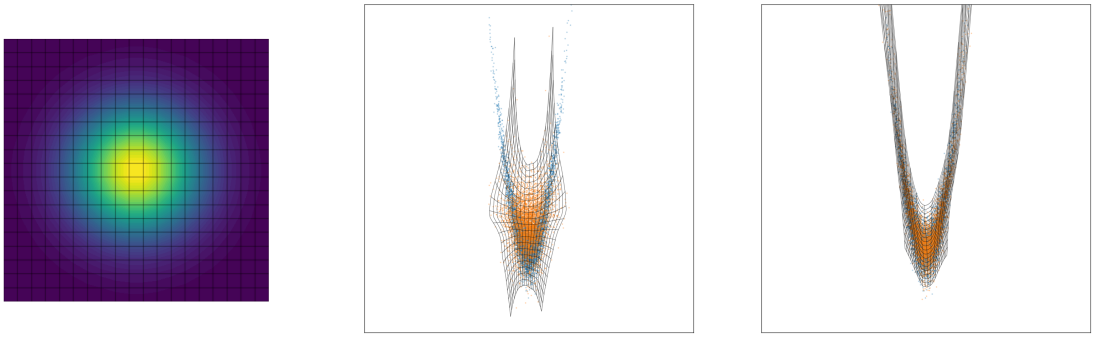


Figure 4.4: Left: The base distribution used by the normalising flow, with an overlaid grid. Middle: The normalising flow at an early stage of training on the banana distribution, where the grid begins to deform toward the target. Right: The normalising flow at the final stage of training, having constructed a close approximation of the target distribution.

The transformation M can be constructed in various ways, provided it remains differentiable and invertible. Several approaches have been proposed in the literature, including RealNVP [4], NICE [3] and MADE [6]. In this thesis, RealNVP is employed, chosen for its simplicity and balance between computational cost and expressive power.

4.3.2 RealNVP

RealNVP defines the transformation M as an affine transformation. Let $z \in \mathbb{R}^N$, and split its dimensions at index n (with $1 \leq n < N$) into $z_{1:n} \in \mathbb{R}^n$ and $z_{n+1:N} \in \mathbb{R}^{N-n}$. The affine transformation uses a scale function $s : \mathbb{R}^{N-n} \to \mathbb{R}^n$ and a translation function $t : \mathbb{R}^{N-n} \to \mathbb{R}^n$ to transform $z_{1:n}$, while leaving $z_{n+1:N}$ unchanged. This construction keeps M invertible, since one part of the input is used to transform the other, and the transformation can therefore also be computed in reverse. The forward transformation is written as

$$\begin{aligned} z^*_{1:n} &= t(z_{n+1:N}) + s(z_{n+1:N}) \odot z_{1:n}, \\ z^*_{n+1:N} &= z_{n+1:N}, \end{aligned} \tag{4.3}$$

where $\odot$ denotes point-wise multiplication. A consequence of the affine transformation is that the Jacobian of (4.3) takes the form

$$J_M = \frac{\partial(z^*_{1:n}, z^*_{n+1:N})}{\partial(z_{1:n}, z_{n+1:N})} = \begin{pmatrix} \operatorname{diag}(s(z_{n+1:N})) & \dfrac{\partial z^*_{1:n}}{\partial z_{n+1:N}} \\ \mathbf{0} & \mathbf{I} \end{pmatrix},$$

where the point-wise multiplication yields the diagonal matrix, and the identity matrix results from the unchanged part $z^*_{n+1:N} = z_{n+1:N}$. Because J_M is lower block-triangular, its determinant depends only on the diagonal blocks and can be expressed as

$$\det(J_M) = \prod_{i=1}^{n} s_i(z_{n+1:N}),$$

where s_i denotes the i-th component of s, and the product arises from the diagonal structure of J_M.

The absolute value required in (4.2) can be accounted for by replacing s with $\exp(s)$, which always produces positive values regardless of the sign of s. Since s does not itself need to be differentiable, it can take the form of a neural network, and the same holds for t. Once parametrised in this way, we denote the resulting neural network-based functions as s_θ and t_θ, so that M becomes a function parametrised by the network weights θ.

To transform all dimensions, several such coupling layers are stacked, with each layer $M^{(k)}$, $k = 1, \ldots, K$, applying its own affine coupling transformation with functions $s^{(k)}_\theta$ and $t^{(k)}_\theta$. Between layers, the roles of the transformed and untransformed partitions are alternated, so that, for example, the first layer transforms $z_{1:n}$ conditioned on $z_{n+1:N}$, and the next transforms $z_{n+1:N}$ conditioned on the updated $z_{1:n}$. The full transformation is then the composition $M = M^{(1)} \circ \cdots \circ M^{(K)}$. An illustration of this can be seen in Figure 4.5. Since the Jacobian of a composition of functions is the product of the individual Jacobians, the log-determinant terms accumulate across layers, giving

$$\log p_X(x) = \log p_Z(z) - \sum_{k=1}^{K} \sum_{i=1}^{n} \exp(s_i^{(k)}),$$

where n is the number of transformed dimensions within each coupling layer, and K is the number of coupling layers composing M.

4.3.3 Conditional normalising flow

So far, the normalising flow has been presented as a way to approximate a distribution $p_X(x)$ by transforming samples from a base distribution $p_Z(z)$ through M. We now extend this to approximate a conditional distribution $p_X(x \mid y)$, where $y \in \mathbb{R}^m$ is an observed conditioning variable. The conditioning is incorporated directly into the transformation, so that M becomes a function of both z and y,

$$M : \mathbb{R}^N \times \mathbb{R}^m \to \mathbb{R}^N,$$

denoted $x = M(z \mid y)$, where $z \sim p_Z(z)$ is still sampled from the base distribution and y enters as an additional input rather than a transformed variable. Since y is not part of the state that is transformed, the invertibility of M and the block structure of its Jacobian are unaffected by the conditioning; only the values of s_θ

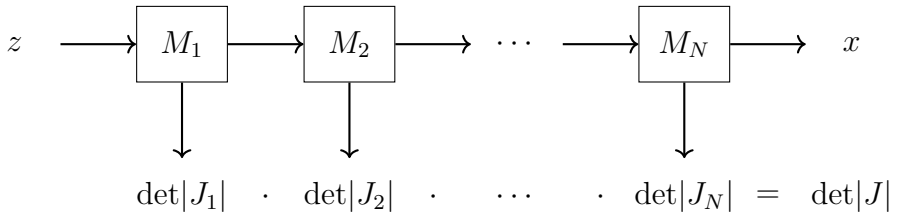


Figure 4.5: Diagram illustrating the transport of point z from the distribution p_z to point x in the distribution $p_x(x)$. The diagram also illustrates how the Jacobian from each individual transformation is summed to obtain the total Jacobian.

and t_θ now also depend on y.

For the RealNVP architecture, this is realised by concatenating the condition y with the untransformed partition, so that the neural networks s_θ and t_θ take $[z_{n+1:N}, y]$ as input instead of $z_{n+1:N}$ alone,

$$s_\theta = s_\theta(z_{n+1:N} \mid y), \qquad t_\theta = t_\theta(z_{n+1:N} \mid y).$$

The conditioning variable y itself is never transformed; it is instead passed as an additional input to every coupling layer $M^{(k)}$, $k = 1, \ldots, K$, so that each layer's scale and translation networks $s_\theta^{(k)}(\cdot \mid y)$ and $t_\theta^{(k)}(\cdot \mid y)$ are conditioned on the same y.

The conditioning now allows us to approximate different distributions depending on the conditional input. It also extends the learning of the correlations of how the change in conditional input changes the output distribution.

4.4 Training

Training refers to the process by which a model's parameters are adjusted to obtain the best possible approximation. Given an optimisation objective, also called a loss function, the quality of the current parameter values can be evaluated and iteratively improved. Neural networks are typically trained using gradient descent, which seeks the minimum of the loss function and thereby the optimal parameters. This section presents *Stochastic Gradient Descent* (SGD), a gradient descent method that uses batches of data to escape local minima.

4.4.1 Stochastic gradient descent

Gradient descent (GD) is an iterative algorithm for finding the minimum of a function. The algorithm proceeds by repeatedly moving in the direction of steepest descent, the negative gradient, under the assumption that this leads towards a minimum. For neural networks, the gradient is computed with respect to the model parameters θ and the loss function $\mathcal{L}$. The training objective is to find the optimal parameters

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta),$$

which is achieved iteratively via the update rule

$$\theta_{k+1} = \theta_k - \eta_k \nabla_\theta \mathcal{L}(\theta_k),$$

where $\eta_k > 0$ is the learning rate, controlling the step size at each update, and $\nabla_\theta \mathcal{L}(\theta_k)$ is the gradient of the loss with respect to the parameters at iteration k. When the loss is large, the gradient is typically large and the parameter update is correspondingly large. When the loss decreases and approaches a minimum, the gradient diminishes and the updates become smaller.

Standard GD computes the gradient using the entire training dataset at each iteration, which makes the optimisation surface smooth but increases the risk of converging to a local minimum. To mitigate this, *Stochastic Gradient Descent* (SGD) replaces the full-data gradient with an estimate computed on a randomly sampled subset of the data, called a batch. The stochasticity introduced by batch sampling causes the optimisation trajectory to fluctuate, which helps the algorithm escape shallow local minima. A further practical benefit is that when the training set is large or effectively unbounded, as is the case when sampling from an SSM, batches are the only computationally feasible approach to training.

A widely used extension of SGD is the *Adaptive Moment Estimation* (Adam) optimiser [13], which uses running estimates of both the first moment (mean) and second moment (uncentred variance) of the gradients. By adapting the learning rate for each parameter, Adam typically achieves faster convergence and greater robustness to the choice of learning rate than standard SGD.

5
Method

The purpose of this chapter is to present the Sequential Normalising Flow (SNF) in detail and specify the test cases used to evaluate it, with Figure 5.1 providing an overview of the workflow. The chapter begins by introducing two test cases, case A and case B, where case A is a linear Gaussian model and case B is a nonlinear Gaussian model. The sampling procedures for both models are then discussed, followed by preprocessing, which describes how samples are normalised to obtain numerically favourable values. Attention then turns to the SNF itself, which is used to approximate the smoothing distribution for the two cases. Its architecture consists of two parts: embedding and distribution construction. The embedding is described first: it encodes future observations into a fixed-dimensional representation, which is then passed to individual normalising flows to obtain the smoothing distribution. Finally, the training procedure is described, including the training setup and hyperparameters used to optimise the SNF parameters.

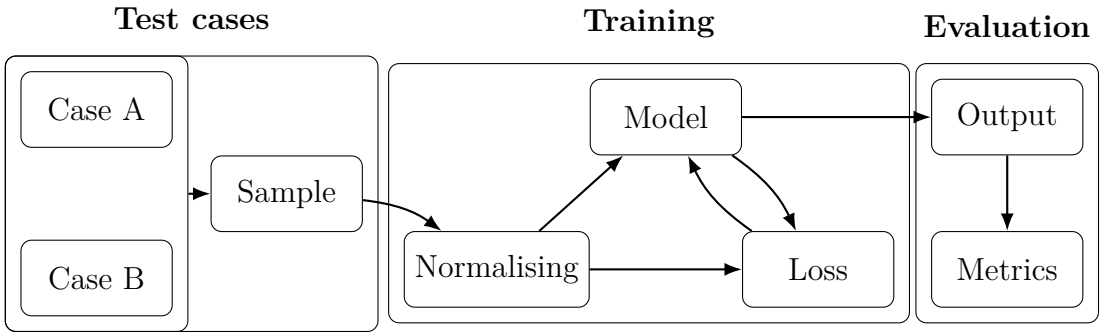


Figure 5.1: Illustrated overview of the method. Left: test cases and sampling. Middle: normalisation of sampled data, SNF model and optimisation objective. Right: after training, the model's outputs are evaluated using a metric.

5.1 Test cases

We now introduce the two test cases used in this thesis. The first is a linear Gaussian model (Case A), chosen as a verification tool to assess the SNF. The second model is non-linear Gaussian model (Case B) and presents a more complex inference problem. For each case, the transition and measurement functions are presented, along with the parameter values used to sample.

5.1.1 Case A

The linear model represents a coupled oscillation model of position and velocity. The model describes a 4-dimensional hidden state that is described as

$$\mathbf{x}_t = \begin{bmatrix} u_x \\ u_z \\ v_x \\ v_z \end{bmatrix}$$

where $u_x, u_z \in \mathbb{R}$ are the positions and $v_x, v_z \in \mathbb{R}$ are the velocities. The hidden state evolves according to

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \epsilon_t$$

where the matrix $\mathbf{A}$ is defined as

$$\mathbf{A} = \begin{bmatrix} 0.95 & 0.05 & 0.20 & 0.00 \\ 0.04 & 0.90 & 0.00 & 0.25 \\ -0.20 & 0.00 & 0.95 & 0.03 \\ 0.00 & -0.25 & -0.02 & 0.90 \end{bmatrix}$$

where positions are updated with respect to themselves, their corresponding velocities, and the other position. The matrix yields a linear transition. The transition noise ϵ_t is sampled as

$$\epsilon_t \sim \mathcal{N}(0, \mathbf{I}_4),$$

where $\mathbf{I}_4$ denotes the 4×4 identity matrix.

The measurement function is a full-state observation. The measurement is obtained via

$$\mathbf{y}_t = \mathbf{B}\mathbf{x}_t + \epsilon_o$$

where the matrix $\mathbf{B}$ is defined as

$$\mathbf{B} = \mathbf{I}_4.$$

The measurement noise ϵ_o is sampled from

$$\epsilon_o \sim \mathcal{N}(0, \mathbf{I}_4).$$

5.1.2 Case B

The nonlinear model is chosen to be a simplified version of a ballistic missile defined in a 2D Cartesian coordinate system, where the two axes represent height and horizontal distance, denoted $u_x, u_z \in \mathbb{R}$ respectively, describing the position of the missile. The velocity components along the same axes are denoted $v_x, v_z \in \mathbb{R}$. Together, these define the state of the ballistic missile at the discrete time t as

$$\mathbf{x}_t = \begin{bmatrix} \mathbf{u}_t \\ \mathbf{v}_t \end{bmatrix} = \begin{bmatrix} u_x \\ u_z \\ v_x \\ v_z \end{bmatrix}$$

The position and velocity are updated separately. Each transition advances the state forward by a time step $\Delta t = 1$, according to the transition function

$$\begin{aligned} \mathbf{u}_{t+1} &= \mathbf{u}_t + \mathbf{v}_t \, \Delta t \\ \mathbf{v}_{t+1} &= \mathbf{v}_t + (-g\mathbf{z} - C(\|\mathbf{v}_t\|)\mathbf{v}_t)\Delta t \end{aligned}$$

where g is the gravitational acceleration, $\mathbf{z}$ is the unit vector in the vertical direction, and $C(\|\mathbf{v}_t\|)$ is the drag coefficient, given as a function of the missile speed $\|\mathbf{v}_t\|$. The drag term is the source of nonlinearity in the transition function. The drag coefficient is given by

$$C(v) = \begin{cases} C_0 & \text{if } v < M \\ C_0 + (C_1 - C_0)(\frac{v-M}{0.1M}) & \text{if } M \leq v \leq 1.1M \\ C_1 e^{-\gamma(\frac{v}{M} - 1.1)} & \text{if } v > 1.1M \end{cases}$$

where the constants are defined as $C_0 = \frac{2}{600^2}$, $C_1 = \frac{6}{600^2}$, $M = 344$ ms^{-1} and $\gamma = 0.3$. The drag model is intended to capture the air resistance experienced at speeds around and above the speed of sound. A realisation of the drag function with these parameter values is shown in Figure 5.2.

The transition noise is added to the velocity and propagates into position through integration, coupling the two via a structured covariance matrix. The transition function with additive noise is written as

$$\begin{pmatrix} u_{t+1} \\ v_{t+1} \end{pmatrix} = f \begin{pmatrix} u_t \\ v_t \end{pmatrix} + \epsilon \, , \quad \epsilon \sim \mathcal{N} \left(\mathbf{0}_4, \begin{bmatrix} \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & 0 & \Delta t & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 & \Delta t \end{bmatrix} \sigma_t^2 \right)$$

where σ_t^2 is the transition noise scale.

The measurement function h is modelled as a ground-based, stationary radar that measures three channels: the range to the target, the radial velocity, and the elevation angle. Assuming the radar is placed at the origin of the coordinate system, the measurement function is given by

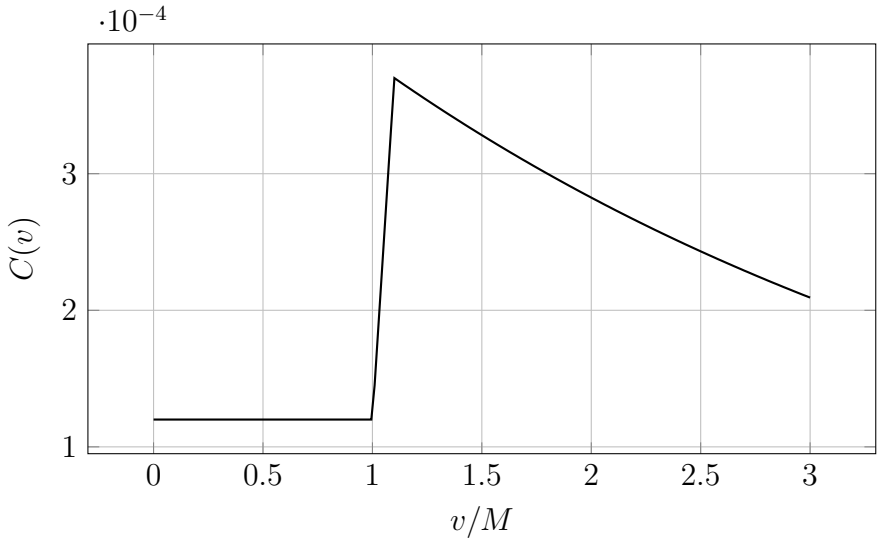


Figure 5.2: Graph showing the function $C(v)$.

$$h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{u_x^2 + u_z^2} \\ \frac{u_x v_x + u_z v_z}{\sqrt{u_x^2 + u_z^2}} \\ \arctan\left(\frac{u_z}{u_x}\right) \end{pmatrix} = \begin{pmatrix} r \\ \dot{r} \\ \alpha \end{pmatrix},$$

where r is the range, $\dot{r}$ is the radial velocity, and α is the elevation angle. Figure 5.3 illustrates both the placement of the radar and the three quantities it measures. The nonlinearity stems from the square roots and inner products in the range and radial velocity terms, and since the mapping from the four-dimensional state to the three-dimensional observation is not invertible, the velocity components v_x and v_z are not directly observed, only their projection onto the range vector.

The measurement noise is modelled as a multivariate Gaussian with a diagonal covariance matrix, since the three channels are assumed independent:

$$R = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_{\dot{r}}^2 & 0 \\ 0 & 0 & \sigma_\alpha^2 \end{bmatrix},$$

where σ_r^2, $\sigma_{\dot{r}}^2$, and σ_α^2 are the variances of the range, radial velocity, and elevation angle measurements. The variances used for both the transition noise and measurement noise are shown in Table 5.1.

5.1.3 Sampling

The two cases, case A and case B, are sampled according to the same procedure but with different parameters. A trajectory begins by sampling the initial state $\mathbf{x}_1$ from a prior distribution p_1, after which the SSM generates the remaining states $\mathbf{x}_{2:T}$ and the corresponding observations $\mathbf{y}_{1:T}$ sequentially according to the transition and

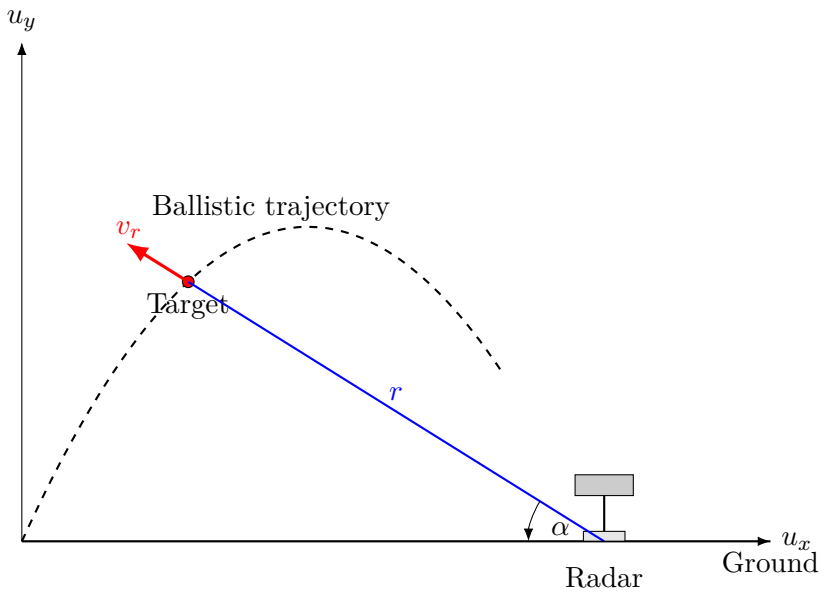


Figure 5.3: Ballistic trajectory observed by radar.

Table 5.1: Transition and measurement noise with their corresponding physical uncertainties.

Parameter	**Realistic representation**
Transition (σ_t)	$\pm$7.1 m/s
Range (σ_r)	$\pm$10 m
Radial velocity ($\sigma_{\dot{r}}$)	$\pm$3.2 m/s
Angle from ground (σ_α)	$\pm$2.2°

measurement functions defined above. For both cases, the trajectories are generated for 25 steps. The prior p_1 is a multivariate Gaussian with a diagonal covariance matrix, reflecting the assumption that the initial state components are independent:

$$\mathbf{x}_1 \sim p_1 = \mathcal{N}(\mu_0, \Sigma_0),$$

where μ_0 and Σ_0 are the initial mean and diagonal covariance matrix. The values used for each SSM are shown in Table 5.2.

Table 5.2: Sampling distribution p_1 for case A and case B.

	Case A		**Case B**	
Dimension	**Mean**	**Std**	**Mean**	**Std**
u_x	0	1	0	10 m
u_z	0	1	0	10 m
v_x	3	3	400	10 m/s
v_z	3	3	400	10 m/s

5.2 Preprocessing

Before training the SNF, the observation data is preprocessed to obtain more favourable values for the model. Large magnitude differences across measurement channels can lead to unstable training and slow convergence, which is particularly relevant to the radar measurement function, as it yields large range values but small elevation angles. To address this, each dimension of the measurement vector is normalised independently using *Z-score normalisation*. Prior to training, a large set of trajectories is sampled from the SSM, and for each measurement dimension i a single mean and standard deviation are computed by pooling the values across all timesteps and trajectories:

$$\mu^i = \mathbb{E}[y_t^i], \qquad \sigma^i = \sqrt{\mathrm{Var}[y_t^i]},$$

where y_t^i denotes the i-th dimension of the observation at time t, and the expectation and variance are taken over all sampled timesteps and trajectories. The resulting statistics μ^i and σ^i are then applied uniformly across all timesteps to normalise the raw measurements y_t^i to $\hat{y}_t^i$, and to invert the normalisation when needed:

$$\hat{y}_t^i = \frac{y_t^i - \mu^i}{\sigma^i}, \qquad y_t^i = \hat{y}_t^i \, \sigma^i + \mu^i.$$

5.3 Sequential Normalising Flow

We now present the Sequential Normalising Flow (SNF), a model that approximates the full smoothing distribution $p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})$. The SNF is constructed from two main components: the embedding of future observations and the generation of the approximating distribution. The embedding transforms observation sequences of varying length into a fixed-size representation, which is then used to condition a normalising flow that generates the approximate distribution.

Approximating the smoothing distribution with a single model across the entire state sequence is computationally expensive and prevents the observation sequence length from varying. The SNF therefore factorises the approximation, exploiting the Markov property of the state-space model so that each factor depends only on the previous state and the future observations:

$$p(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T}) \approx q_{\phi_1}(\mathbf{x}_1 \mid \mathbf{y}_{1:T}) \prod_{t=2}^{T} q_{\phi_t}(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{y}_{t:T}).$$

An overview of this architecture is shown in Figure 5.4.

We now proceed to present the embedding procedure for the future observations, followed by how the SNF architecture generates the approximating distribution.

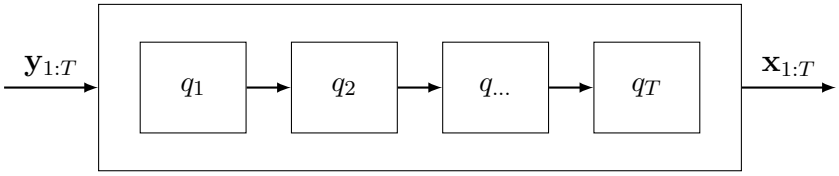


Figure 5.4: Overview of the SNF architecture. Future observations $\mathbf{y}_{1:T}$ enter the model, which sequentially generates the state sequence $\mathbf{x}_{1:T}$ through the chain of conditional distributions $q_1, \ldots, q_T$.

5.3.1 Embedding observations

The SNF is viewed as two components: an embedding model and a generative model. The embedding model's purpose is to represent the future observations $\mathbf{y}_{t:T}$ as a fixed-dimensional representation, since the generative model requires fixed-size inputs regardless of how many future observations remain. This embedding is realised using a GRU, presented in Section 4.2.

A property of the GRU is that it represents information from the end of the sequence more strongly than information from the beginning. Since the most recent observations are the most informative for estimating the current state, the observation sequence is reversed from $\mathbf{y}_{1:T}$ to $\mathbf{y}_{T:1}$ before being passed to the GRU. The GRU then outputs an encoded representation e_t for each time step, and the resulting sequence of hidden states is reversed back to restore the correct temporal order:

$$e_{1:T} = \text{rev}(\text{GRU}(\mathbf{y}_{T:1})),$$

where $\text{rev}(\cdot)$ denotes reversing the order of the hidden state sequence. This ensures that e_t, the embedding used at time step t, places the strongest emphasis on the observations closest to t. For notational convenience, this reversal is left implicit in the remainder of the thesis, and the embedding is instead written as

$$e_{1:T} = \text{GRU}(\mathbf{y}_{1:T}).$$

The GRU is constructed with 25 individual units, matching the sample sequence length T. Each hidden state e_t is a 256-dimensional vector, and the input dimension is either 4 or 3, corresponding to the dimensionality of $\mathbf{y}_t$ in case A and case B, respectively.

5.3.2 Sampling trajectories

Each individual unit of the SNF follows the RealNVP architecture, as presented in Section 4.3.2, with the number of layers per unit, denoted k, held constant across all units. The architecture of the scaling network s_θ and the translation network t_θ, which together parameterise each flow, is presented in Table 5.3.

Figure 5.5 illustrates the inputs and outputs of both the SNF (top row) and the GRU embedding model (bottom row). Since the final normalising flow requires the output of the previous one, the full observation sequence must be encoded by the

Table 5.3: Layer-wise architecture of the scaling network s_θ and the translation network t_θ.

Layer	s_θ		t_θ	
	Type	**Output Shape**	**Type**	**Output Shape**
1	Input	-	Input	-
2	Linear	256	Linear	256
3	LeakyReLU	256	LeakyReLU	256
4	Linear	256	Linear	256
5	Tanh	4	Output	4

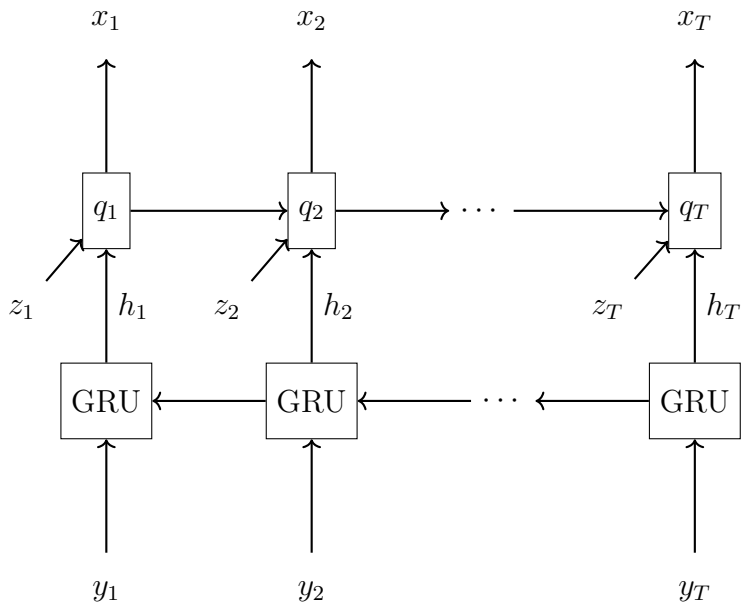


Figure 5.5: GRU-conditioned sequential normalising flow with latent variables.

GRU before any samples from the SNF can be generated.

Rather than letting each conditional distribution q_t generate the next state $\mathbf{x}_t$ directly, q_t instead generates a residual r_x relative to the prior mean $f(\mathbf{x}_{t-1})$, obtained by propagating the previous state through the transition function f. The state is then recovered as

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}) + r_x, \qquad r_x \sim q_t.$$

The model therefore learns to correct the dynamical prior rather than generate the state from scratch.

Each q_t is conditioned on three inputs at time t: the embedding e_t, obtained as described in Section 5.3.1; the normalised observation $\hat{\mathbf{y}}_t$; and the residual between the predicted and observed measurement,

$$\hat{h}\Big(f(\mathbf{x}_{t-1})\Big) - \hat{\mathbf{y}}_t,$$

where the predicted measurement $h\big(f(\mathbf{x}_{t-1})\big)$ is normalised, denoted $\hat{h}$, to match the scale of $\hat{\mathbf{y}}_t$. This gives

$$r_x \sim q_t\Big(\cdot \mid \hat{h}\big(f(\mathbf{x}_{t-1})\big) - \hat{\mathbf{y}}_t,\ \hat{\mathbf{y}}_t,\ e_t\Big)\,.$$

The first unit, q_1, has no previous state to condition on and is therefore conditioned only on the embedding and the first observation. To keep its structure consistent with the remaining units, its residual is instead defined around the mean μ_0 of the prior distribution p_1, presented in Section 5.1.3:

$$\begin{aligned}\mathbf{x}_1 &= \mu_0 + r_x,\\ r_x &\sim q_1(\,\cdot \mid \hat{\mathbf{y}}_1,\ e_1)\,.\end{aligned}$$

By defining each unit's output as a residual relative to a state-dependent prior, rather than as an absolute state, every q_t solves the same local problem regardless of t: correcting a predicted mean given the current observation and embedding. This allows the same architecture, and the same parameters θ, to be reused recursively across all time steps, rather than requiring a separate model for each. At each time step, q_t thus defines a distribution over r_x, from which a sample is drawn and added to the prior mean to obtain $\mathbf{x}_t$; this state is then propagated forward through f to condition q_{t+1}, and the process repeats until the full trajectory $\mathbf{x}_{1:T}$ has been sampled. The joint density $q(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})$ used during training follows directly from the factorisation over $q_1, \ldots, q_T$.

5.4 Training

Training the SNF begins by sampling from the selected SSM, as described in Section 5.1.3. The sampled observations are propagated through the SNF model, and the parameters are then updated according to the optimisation objective. To update the parameters, the ADAM optimiser, presented in Section 4.4.1, is used.

The optimisation objective used to train the SNF is the ELBO, described in Section 3.3. With the ELBO, batch training is used. For each update, a batch of N individual observation sets is sampled. In PyTorch, the optimiser minimises the given optimisation function. Therefore, the ELBO is rewritten from maximising to minimising and takes the form

$$\begin{aligned}\mathcal{L}(\phi) = \frac{1}{N}\sum_{n=1}^{N}\Bigg(&\log\, q_\phi(x_1|y_{1:T}^{(n)}) + \sum_{t=2}^{T}\log\, q_\phi(x_t|x_{t-1}, y_{i:T}^{(n)}) - \\ &\log\, p(x_1) - \sum_{t=2}^{T}\log\, p(x_t|x_{t-1}) - \sum_{t=1}^{T}\log\, p(y_t^{(n)}|x_t)\Bigg)\end{aligned}$$

where t is the timestep and n is the index of the measurement set in the batch. The latent state is obtained by $x_{1:T} \sim q_\phi(y_{1:T})$. Training of the model is performed until convergence is reached.

Additional hyperparameters are introduced to stabilise training. A learning rate scheduler is used. The learning rate scheduler monitors the optimisation objective's value and, if it has not decreased over several updates, reduces the learning rate. The reduction consists of multiplying the active learning rate by 0.8.

6
Results

This chapter presents the results of the SNF's ability to approximate the smoothing distribution. It begins by describing the evaluation metrics, and thereafter, the SNF is evaluated on the two test cases from Chapter 5.

6.1 Evaluation

The performance of the SNF is evaluated using three metrics. These are the Mean Squared Error (MSE), the Kullback–Leibler Divergence, and a visual interpretation of the dimensional reduction using t-SNE.

6.1.1 Mean square error (MSE)

In this thesis, the MSE is used in two ways. The first compares, at each time step t, the mean of the SNF's approximating distribution $\boldsymbol{\mu}_t^{\text{SNF}}$ to the mean produced by the Kalman smoother, $\boldsymbol{\mu}_t^{\text{KS}}$:

$$\text{MSE}_t^{\text{SNF-KS}} = (\boldsymbol{\mu}_t^{\text{SNF}} - \boldsymbol{\mu}_t^{\text{KS}})^2.$$

For the linear Gaussian case, the Kalman smoother is optimal, so this quantity is expected to approach zero as the SNF converges to the true smoothing distribution.

The second compares, at each time step t, the mean of the Kalman smoother and the SNF to the true hidden state $\mathbf{x}_t$:

$$\text{MSE}_t^{\text{KS-true}} = (\boldsymbol{\mu}_t^{\text{KS}} - \mathbf{x}_t)^2, \qquad \text{MSE}_t^{\text{SNF-true}} = (\boldsymbol{\mu}_t^{\text{SNF}} - \mathbf{x}_t)^2.$$

Unlike the linear Gaussian case, the Unscented Kalman Smoother is not optimal for the nonlinear ballistic case, so its mean is not guaranteed to lie closer to the true state than the SNF's. Comparing $\text{MSE}_t^{\text{KS-true}}$ and $\text{MSE}_t^{\text{SNF-true}}$ therefore provides a way to assess whether the SNF approximates the true hidden state better or worse than the Unscented Kalman Smoother.

6.1.2 Kullback–Leibler Divergence

The second evaluation metric is the Kullback–Leibler (KL) divergence, introduced in Section 5.4. Here, it is used as a measure of distributional similarity between the generative model output and the true smoothing distribution. After training, the

KL divergence between the SNF and the joint distribution is computed over a fixed set of observations. The KL divergence is defined as

$$D_{KL}\left(q_\phi(x_{1:T} \mid y_{1:T}) \,||\, p(x_{1:T} \mid y_{1:T})\right) = \\ \mathbb{E}_{x\sim q_\phi}[\log\ q_\phi(x_{1:T} \mid y_{1:T}) - \log\ p(x_{1:T} \mid y_{1:T})] + \underbrace{\log p(y_{1:T})}_{Constant}$$

where $\log p(y_{1:T})$ is constant. The KL divergence yields the logarithmic marginal likelihood if the approximation yields an exact representation. The logarithmic marginal likelihood can be computed via the Kalman filter for linear Gaussian systems, as described in Section 2.7 and provides a tractable reference for evaluating the approximation of the smoothing distribution by the SNF.

6.1.3 T-SNE

The third metric is a visual assessment of the embedding space produced by the GRU. Recall from Section 5.3.1 that, in the SNF, a GRU is used to produce a sequential representation e_t of the information contained in a backward pass of the future measurements. Since this representation is high-dimensional, direct inspection is not feasible. T-Distributed Stochastic Neighbour Embedding (t-SNE) is a dimensionality reduction algorithm that projects a high-dimensional space into two or three dimensions while preserving the local structure between data points, allowing clusters in the high-dimensional space to be identified visually. As t-SNE is an established method used here solely as a visualisation tool, the reader is referred to the original paper for a full description [14].

6.2 Case A

We now present the results for the linear Gaussian case. The setting of this scenario is presented in Section 5.1.1. The following parameters are used when generating samples: transition noise, $\sigma_t = 3$ and measurement noise, $\sigma_m = 3$. In this case, the Kalman smoother serves as a ground-truth reference for the smoothing distribution.

6.2.1 Training

Training of the SNF was performed according to the method presented in Section 5.4. During training, a batch size of 256 was used, with batches generated before each update. Each sample in the batch contains a unique set of observations. A learning rate scheduler was used, which reduces the learning rate by a factor of 0.8 if the loss does not decrease within a window of 25 epochs, after which the window resets and the process repeats. Beyond its role in reducing the learning rate, the scheduler therefore also provides an indication of when the loss has stopped decreasing, since the learning rate is repeatedly decreased whenever the loss fails to improve. A learning rate that keeps decreasing towards the end of training shows that the loss is no longer decreasing by more than a marginal amount. The loss function during

training of the SNF is seen in Figure 6.1. The model was trained several times, and it was concluded that within 500 updates, the model had converged. This is further supported by the learning rate schedule. After 200 updates, the learning rate begins to decrease consistently, indicating that the loss has plateaued.

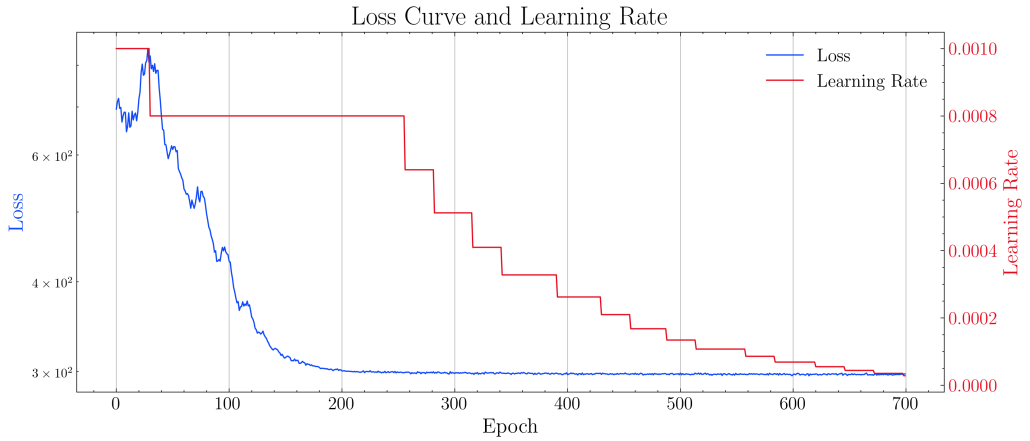


Figure 6.1: Graph showing the objective function (loss) value when training on the linear SSM, illustrated in blue. The red curve represents the change in learning rate during training.

6.2.2 One observation sequense

After training, the SNF was evaluated on a single observation sequence $\mathbf{y}_{1:T}$ sampled from case A, alongside both the KF and KS solutions, which were given to the same sequence. The result is shown in Figure 6.2. The lines show the mean at each timestep and the fill shows the standard deviation. For the KF and KS, this is given from the diagonal of the respective covariance matrix, while for the SNF, it is computed from 500 generated trajectories.

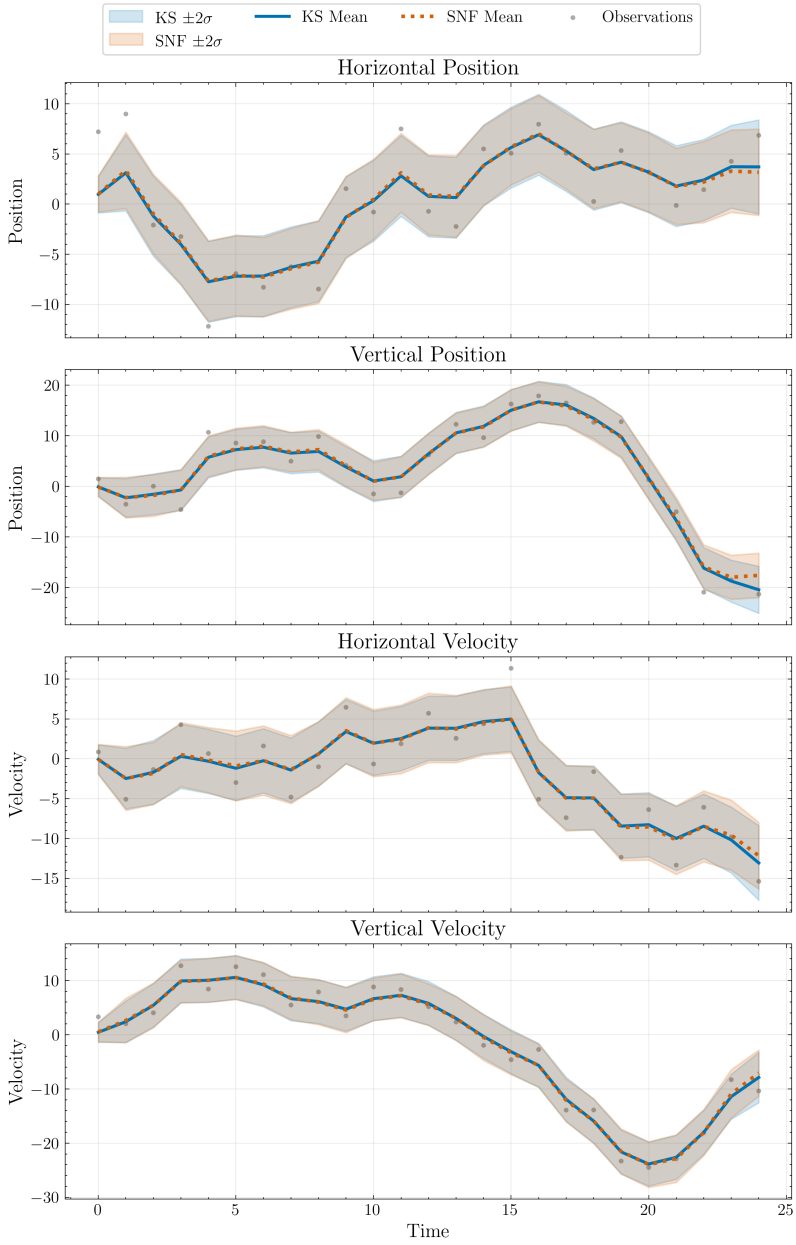


Figure 6.2: Solutions for one sequence for case A. Orange: KF. Blue: KS. Red: SNF.

The KS yields a better estimate of the latent states than the KF. The SNF outputs an almost identical distribution to the KS, with the mean lines overlapping throughout. The only notable deviation occurs near the end of the sequence, visible across all dimensions in both the mean and the standard deviation.

The KS yields an optimal solution in this linear case. Therefore, the SNF should match the KS solution if its approximation of the true smoothing distribution is exact. In contrast, the KF is only optimal given observations up to each respective time, and consequently differs from the KS precisely by the extent to which future observations improve the estimate. Table 6.1 shows the MSE between the true state

from the SSM and each of the three solutions. The SNF achieves an MSE close to that of the KS and clearly below that of the KF, confirming that it successfully incorporates future observations.

Table 6.1: MSE between the true state and the KF, KS, and SNF, for one sequence from case A.

Model	KF	KS	SNF
MSE	6.212	3.920	4.030

The distributional similarity between the SNF and the true smoothing distribution is further assessed by comparing the KL divergence and the resulting marginal likelihood, as shown in Table 6.2. The KL divergence is computed with 5000 samples, and the log marginal likelihood is estimated via the Kalman filter. The resulting ELBO gap is small, indicating that the SNF closely approximates the true smoothing distribution.

Table 6.2: KL divergence, log marginal likelihood, and ELBO gap for one sequence from case A. A perfect approximation gives an ELBO gap of zero.

KL	$\log p(\mathbf{y}_{1:T})$	ELBO gap
302.807	-301.684	1.123

6.2.3 Multiple observation sequences

To understand the SNF's general behaviour when approximating the smoothing distribution, a study was conducted across 25 distinct observation sequences sampled from case A. For each sequence, the SNF generated 500 samples, and the mean of these samples was computed at each time step and compared to the KS mean using the MSE. This comparison between the SNF and KS distribution means at each time step was repeated for every observation sequence, and the resulting errors were averaged across sequences to obtain a single error estimate per time step, shown in Figure 6.3. The line shows this average error between the SNF and KS means at each timestep, and the fill shows the corresponding standard deviation across sequences. The error remains nearly constant, below 0.1, for the first 80% of the time steps, before growing rapidly towards the end: a larger error and standard deviation appear around timestep 20, followed by exponential growth, with the final timestep showing the largest error and standard deviation of all.

This behaviour is thought to arise from how the number of future observations available to the model changes with the timestep. The final timestep, conditioned on only the current observation, is easier to solve than earlier timesteps, which must be estimated from several future observations. We attribute the growing error to how the embedding is constructed and how the normalising flow interprets it. The GRU builds its representation through recurrent updates, so with fewer future observations to process, the hidden state has undergone less transformation

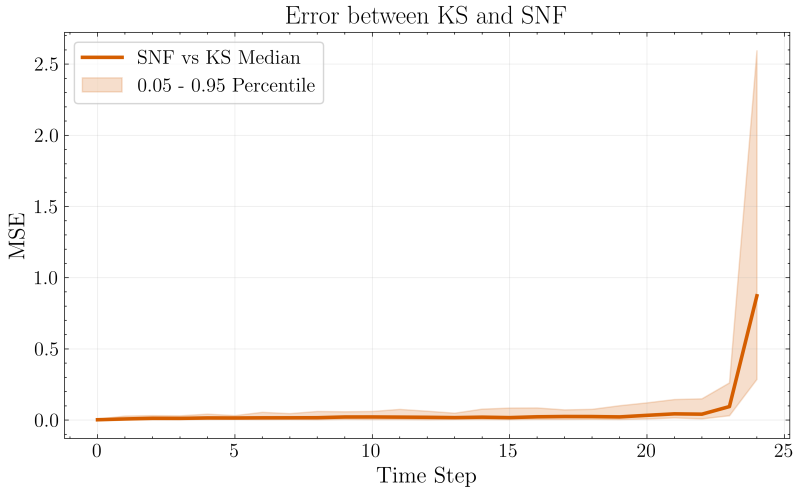


Figure 6.3: Average MSE between the SNF mean and the KS mean at each time step, across 25 independent sequences. The shaded band shows the standard deviation across sequences.

and therefore occupies a different region of the embedding space than it would for a longer sequence. As a result, embeddings produced near the end of the sequence are structurally dissimilar to those produced from longer subsequences, making it difficult for the shared normalising flow unit to generalise across all time steps. Based on these results, the model maintains reliable performance for sequences with at least approximately two remaining future time steps, beyond which the embedding becomes insufficient for the normalising flow to produce accurate samples.

To test this explanation, the GRU was used to embed 256 independent observation sequences, and the resulting 256-dimensional hidden state at each timestep was reduced to two dimensions using t-SNE. The result is shown in Figure 6.4, with points colour-coded by timestep. A cluster can be observed, isolated from the rest of the scatter. Figure 6.5 isolates individual time steps and confirms that this cluster corresponds to the final two time steps, 24 and 25, while all earlier time steps are mixed throughout the rest of the embedding space.

This indicates that which timestep an embedding originates from does not matter, except for the final two timesteps, which occupy a distinct region of the embedding space. This is a desirable property: since the same normalising flow unit is applied recursively across all timesteps, the embedding should ideally be invariant to timestep for the flow to generalise consistently. The final two timesteps break this invariance, forcing the flow to reconcile two different relative distributions within a region that constitutes only a small fraction of the full sequence, which explains why the resulting error is concentrated and disproportionately large there. This suggests that a more temporally consistent embedding would improve the SNF's performance.

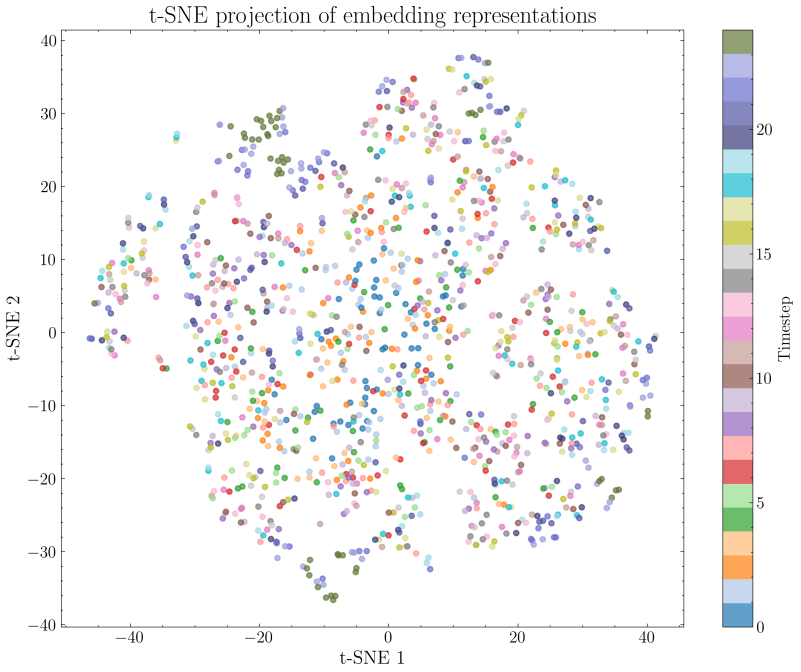


Figure 6.4: Dimensional reduction of the embedding representation for 256 sets of observations using t-SNE. The observations are samples from the linear SSM. Points in the figure are coloured according to the timestep they represent.

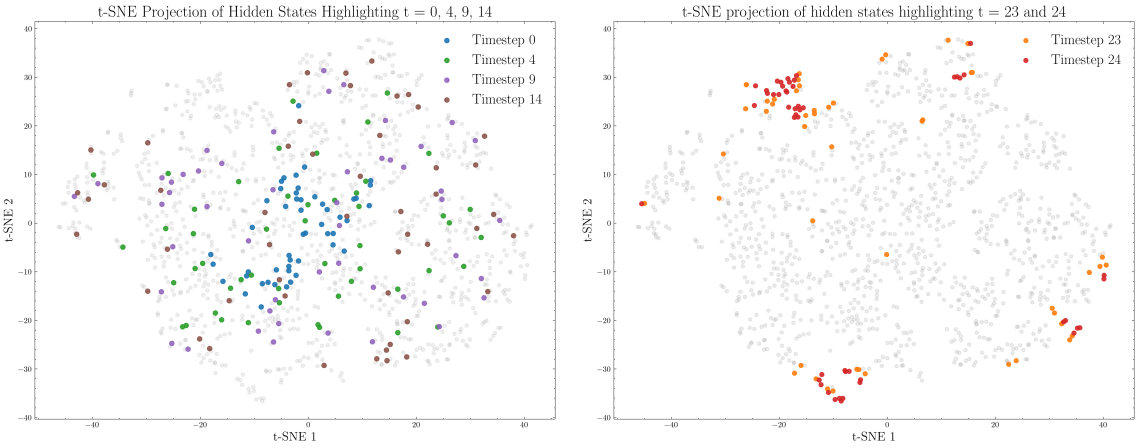


Figure 6.5: Dimensional reduction of the embedding representation for 256 sets of observations using t-SNE. Selected time steps are colour-coded, while the rest are shown in grey. Left: timesteps 1, 5, 10, 15. Right: timesteps 24, 25.

6.3 Case B

The SNF's performance on the ballistic model is now presented. Since the ballistic model has both a nonlinear transition function and a nonlinear measurement function, it poses a more difficult problem than the previous case. The model's training is presented first, followed by evaluation on a single fixed observation sequence and then across several observation sequences. As a reference solution, the UKS is used in place of the KS, since the model is nonlinear. Note that the UKS constrains its prediction to a Gaussian, so unlike the KS in the linear case, its solution is not optimal, but still serves as a good reference approximation.

6.3.1 Training

As in the linear case, a learning rate scheduler was used, reducing the learning rate by a factor of 0.8 if the loss did not decrease within a window of 35 updates the window then resets and the process repeats. The model was concluded to have converged after 2000 iterations, consistent with the scheduler's learning rate decreasing as the loss plateaued. The loss is shown in Figure 6.6. It decreased rapidly at the start and exhibited oscillations while descending to its converged state; these oscillations are believed to arise from training with a small batch size, resulting in a noisier loss signal.

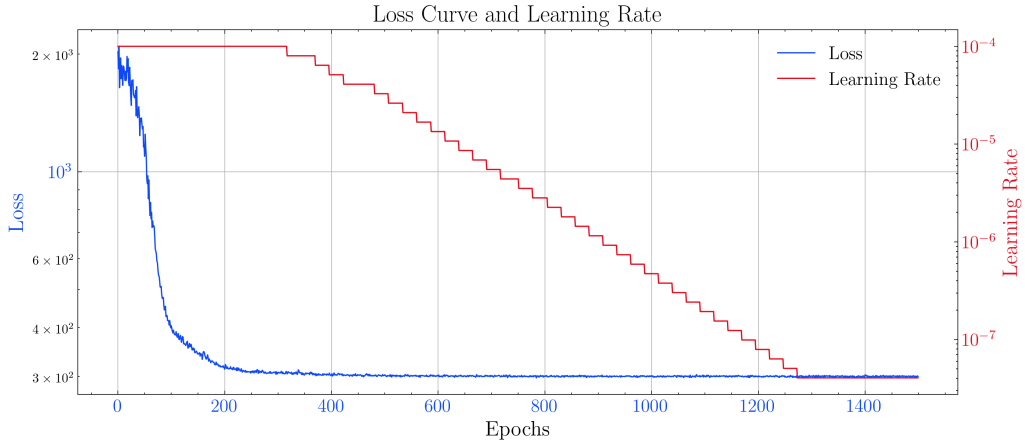
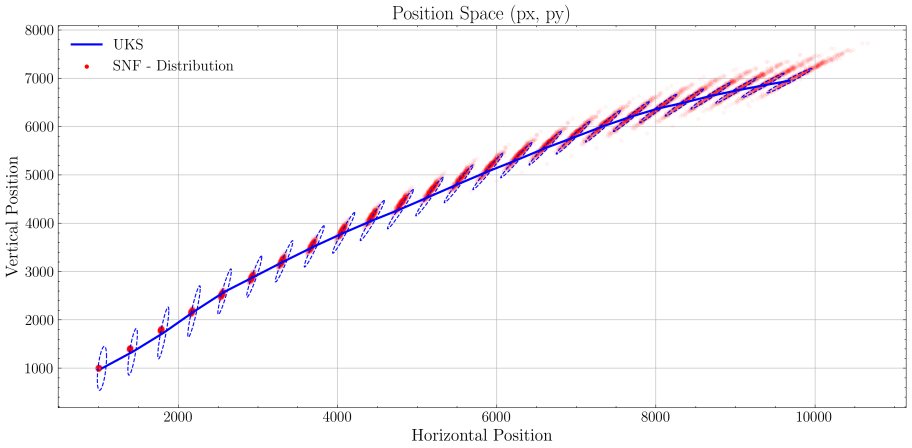


Figure 6.6: The loss function plotted against the update step (left axis), together with the learning rate (right axis).

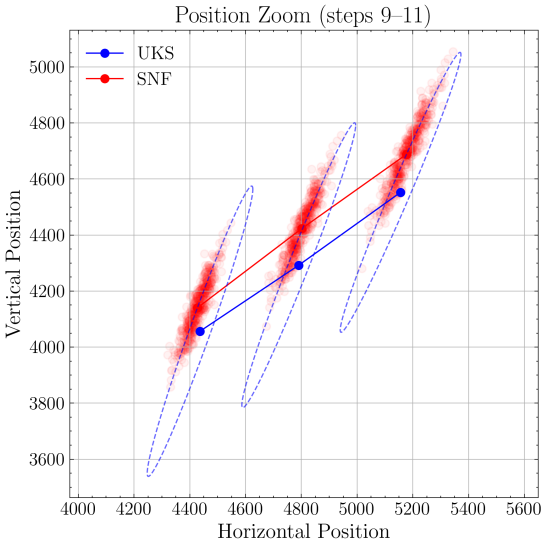
6.3.2 One observation sequence

After training, the SNF was evaluated on a single observation sequence, with the results grouped by position and velocity. Figure 6.7a shows the SNF's position estimates over time, compared with the UKS solution for the same observations. The UKS mean and covariance are shown as a blue line and blue circles, respectively, while the SNF is shown as both a mean and individual samples at each timestep. The shape of the UKS distribution and the spread of the SNF samples both reflect

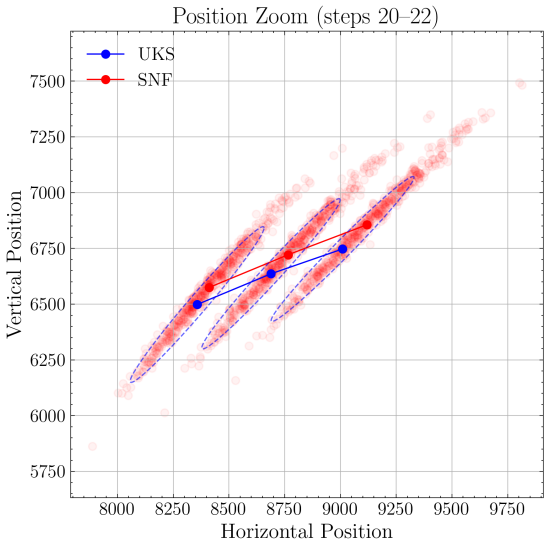
the structure of the measurement function, uncertainty in the angle measurement produces stretched distributions and the distributions rotate over time in a way that reflects the radar's position at $(16000, 0)$. The SNF sample spread is smaller at the beginning of the trajectory than at the end, whereas the UKS distributions remain more consistent in size throughout. The SNF samples do, however, rotate in line with the UKS, indicating that the model has correctly learned the structure of the measurement function.



(a) UKS (blue) and SNF (red) position estimates for one observation sequence.



(b) Time steps 9 to 11.



(c) Time steps 20 to 22.

Figure 6.7: Comparison of UKS and SNF position estimates for the ballistic example.

To examine this similarity more closely, the solutions are compared over the shorter intervals $t = [9, 11]$ and $t = [20, 22]$, shown in Figures 6.7b and 6.7c. Both intervals show the same pattern. The SNF distribution resembles the UKS in shape, but its

mean is offset from the UKS mean. This suggests that the SNF captures the overall shape of the smoothing distribution but struggles to place its samples accurately.

For the velocity dimensions, the SNF and UKS mean estimates are shown in Figure 6.8. The spread is omitted here and instead shown for the smaller intervals below. The two estimates are misaligned throughout the trajectory, with a small offset at the start that grows towards the end, reaching its largest value at the final timestep. The UKS mean also progresses in more evenly spaced steps than the SNF mean, which moves in larger, less regular steps between timesteps. The true latent state is shown in grey for reference.

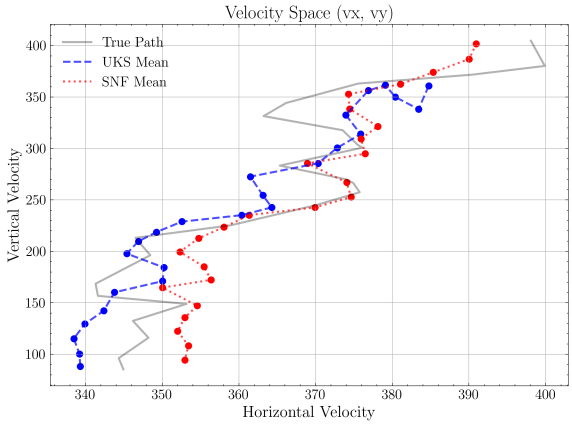


Figure 6.8: UKS (blue) and SNF (red) velocity estimates for one observation sequence.

The same comparison is repeated for velocity over the intervals $t = [9, 11]$ and $t = [20, 22]$, shown in Figures 6.9 and 6.10. Both intervals confirm the misalignment seen in Figure 6.8. The SNF and UKS distributions do not align. Overall, the SNF shows greater difficulty estimating velocity than position, which may be explained by the reduced information available on velocity through the measurement function.

The MSE between the true latent state and the UKF, UKS, and SNF estimates, computed over all state dimensions, is shown in Table 6.3. As in the linear case, comparing the SNF to both the UKF and UKS indicates how well it incorporates future observations. Here, however, the SNF has the highest error of the three models, suggesting that it struggles to correctly interpret the input in this nonlinear setting.

Table 6.3: MSE between the true state and the UKF, UKS, and SNF, for one sequence from case B.

Model	UKF	UKS	SNF
MSE	11137.7	2586.9	27357.4

For this sequence, the KL divergence and marginal likelihood were also computed, shown in Table 6.4. The resulting ELBO gap is larger than in the linear case, con-

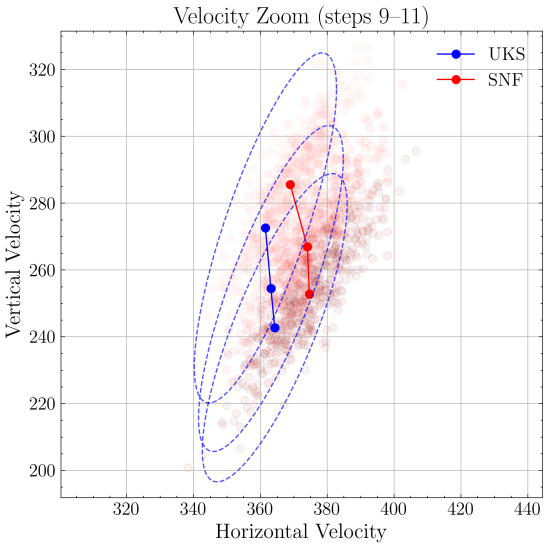


Figure 6.9: Time steps 9 to 11.

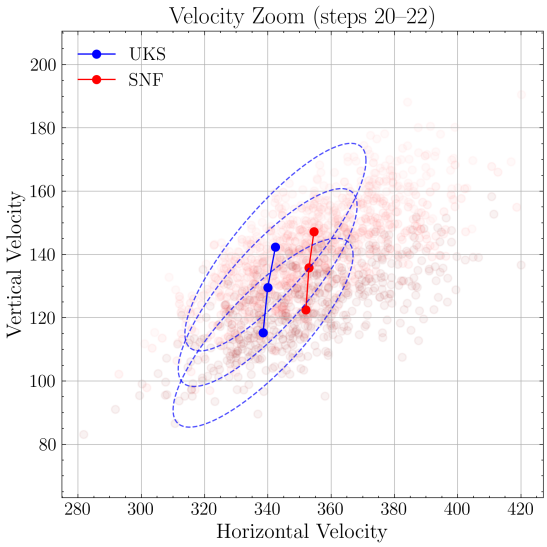


Figure 6.10: Time steps 19 to 21.

sistent with the misalignment observed above and the higher MSE, and indicates a poorer approximation of the smoothing distribution despite the training loss having converged. It should be noted, however, that the UKS is itself only an approximation of the true smoothing distribution, so the MSE comparison above should be interpreted with this in mind.

Table 6.4: KL divergence, log marginal likelihood, and ELBO gap for one sequence from case B.

KL	$\log p(\mathbf{y}_{1:T})$	ELBO gap
313.01	-297.76	15.25

6.3.3 Multiple observation sequenses

To assess the model's general approximation capability, the SNF and UKS were evaluated on 25 distinct observation sequences by comparing the mean of each solution with the true hidden state at each time step. The result is shown in Figure 6.11.

The UKS error remains roughly constant across all time steps, while the SNF error increases over time, similar to the pattern observed for the final time steps in the linear case. This suggests that the SNF's error compounds over time due to earlier, poorer estimates. The model generates distributions with the expected shape, but their locations become increasingly misaligned.

To investigate the cause, the GRU embedding is analysed as in the linear case, 256 observation sequences are sampled and embedded, and the resulting representations are reduced to two dimensions using t-SNE, shown in Figure 6.12. Unlike the linear case, a clear correlation between clusters and timesteps is seen here, indicating that the GRU struggles to produce time-independent representations. Figure 6.13

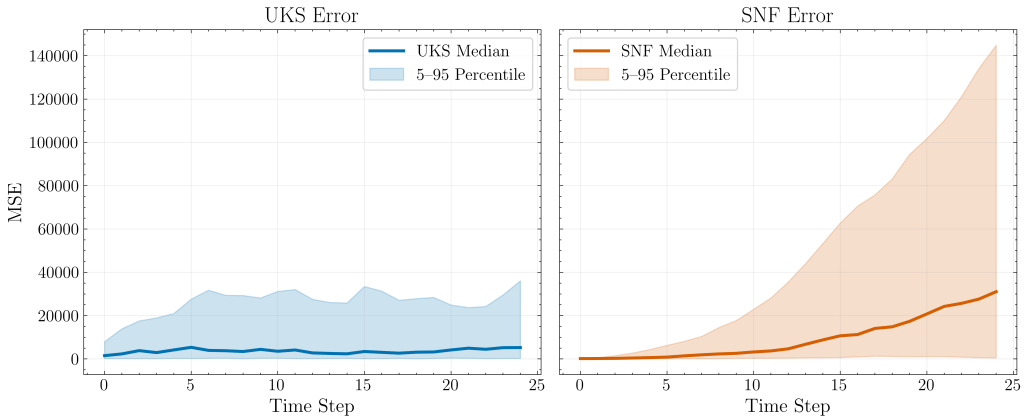


Figure 6.11: MSE to the true latent state. Left: UKS. Right: SNF.

confirms this at the level of individual timesteps. Since the SNF models a residual distribution relative to the dynamic prior, embeddings from different timesteps should be similar whenever the underlying residual distributions are similar. The clustering by timestep, therefore, suggests that the embedding fails to capture this relative structure and, consequently, that the generative model struggles to correctly interpret what a given embedding represents.

6.4 Running time

The running time of the SNF was compared with those of the KS and UKS for both cases, averaged over 256 independent observation sequences. Comparing running times is inherently difficult, as many factors affect computational cost; the results presented here are therefore intended only to give a general sense of the SNF's computational cost relative to the classical smoothers, rather than a rigorous benchmark. All times were obtained with the KS and UKS running on the CPU and the SNF running on the GPU. The KS and UKS times correspond to computing the mean and covariance at each time step, while the SNF time corresponds to sampling 500 trajectories. The results are shown in Table 6.5.

Table 6.5: Mean runtime over 256 sequences, comparing the SNF to the KS (case A) and UKS (case B).

Case	Method	Runtime (seconds)
Case A	Kalman Smoother	0.010586
	Sequential Normalising Flow	0.033051
Case B	Unscented Kalman Smoother	0.346570
	Sequential Normalising Flow	0.147741

In case A, the KS is faster than the SNF, though the difference is modest, and further optimisation of the SNF architecture could plausibly close this gap. In case B, the relationship reverses: the SNF is faster than the UKS. However, since the SNF's

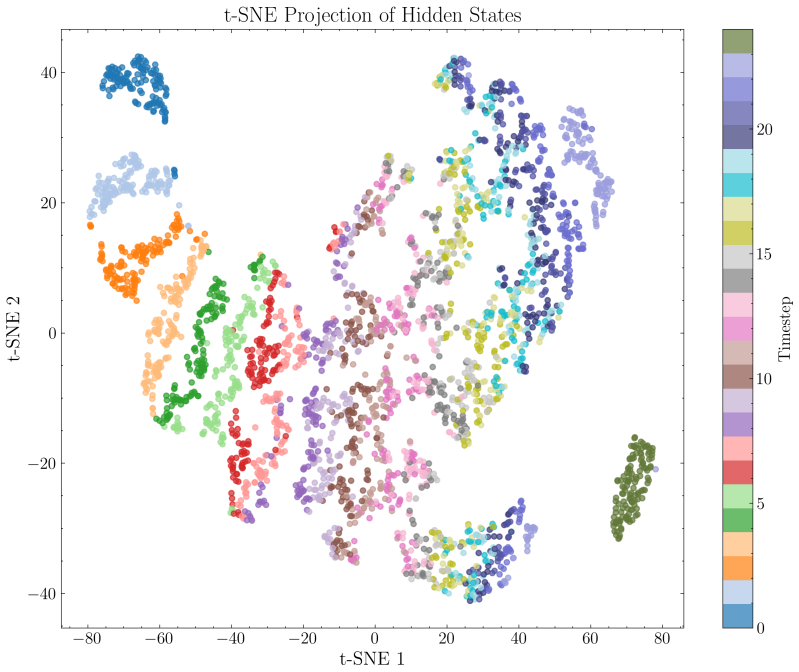


Figure 6.12: t-SNE of the embedding for 256 sequences sampled from the ballistic SSM, coloured by timestep.

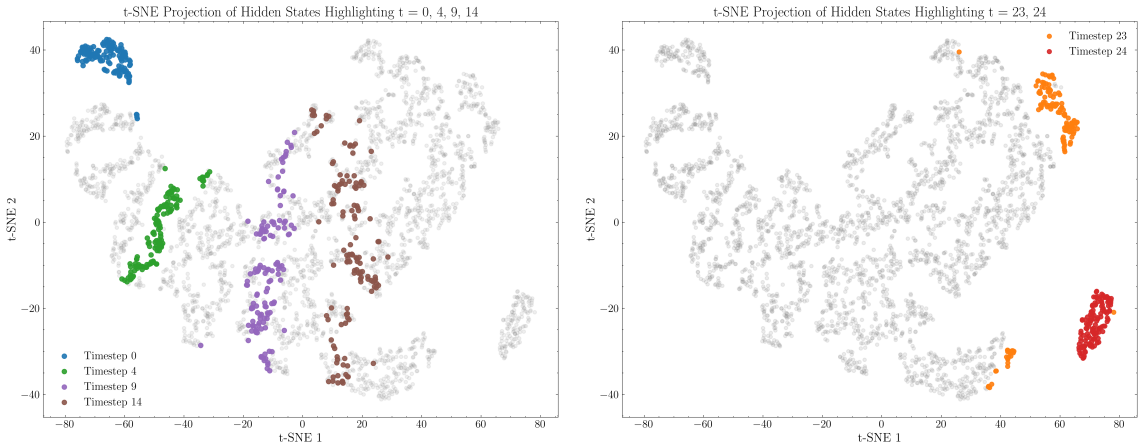


Figure 6.13: Selected timesteps colour-coded, rest in grey. Left: timesteps 1, 5, 10, 15. Right: timesteps 24, 25.

approximation is comparatively poor in case B, as shown in the preceding sections, this speed advantage should not be taken as evidence that the SNF is currently competitive with the UKS overall. It instead suggests that, if the approximation accuracy could be improved without significantly increasing model size, for example through a better embedding, the SNF could become competitive with classical smoothers in both accuracy and computational cost.

7
Discussion

This chapter discusses the findings and their interpretation, connecting the results to existing literature. It begins by examining the factors believed to cause the observed performance, as well as the constraints imposed by the data the model is trained on. This is followed by a discussion of the project's contributions to the research field and how the work could be extended. Finally, the chapter concludes by outlining potential future work and its relevance.

7.1 Aspects of model performance and usage

The SNF performs well in the first case, demonstrating the ability to approximate the smoothing distribution. However, when applied to the more demanding ballistic model, the SNF struggles to achieve an equally good approximation. The model learns some correlation but does not converge to an optimal solution. The SNF's architecture consists of two components: the embedding model and the generative model. The generated distributions in the results demonstrate good expressive capability, for example, in the case of the radar measurement function. This is consistent with the literature, in which the RealNVP architecture has demonstrated strong expressiveness. The normalising flow is therefore not considered a bottleneck in the SNF. The embedding model, on the other hand, appears to have difficulty providing the normalising flow with useful representations. The results clearly indicate that the embedding model struggles to produce time-independent representations, which is believed to, in turn, confuse the generative model during sampling. This is considered the dominant limitation, and further investigation is recommended.

The choice of optimisation objective is another factor believed to affect SNF performance. The training results indicate convergence, with no further improvement beyond a certain number of updates, yet the model does not achieve a perfect approximation. Training is performed by approximating the full smoothing distribution, which yields a 300-dimensional space derived from 25 timesteps, 4 latent-state dimensions, and 3 measurement dimensions. In contrast, the UKS computes the marginal density $p(x_t \mid y_{1:T})$, which operates in a lower-dimensional space. Training the SNF with an objective that targets only the marginal density could reduce the risk of getting stuck in local minima, potentially leading to both better approximations and faster training. However, how this could be implemented is beyond the scope of this thesis.

Considering the use of such a model in a real-world setting, the SNF can only be applied where training data is available and only for scenarios represented in that data. This prevents it from being applied to previously unseen scenarios. Traditional methods, by contrast, can produce a solution for any set of observations drawn from a different distribution, provided the necessary inputs are available. This means that for the SNF to generalise effectively, it either requires access to a very large and diverse dataset or must be trained on dynamics broad enough to cover a wide range of possible transition dynamics.

7.2 Research perspective

The aim of this thesis has been to construct a model capable of approximating the full smoothing distribution, enabling a computationally cheap posterior. The project has demonstrated how to construct and train the SNF such that it can, in principle, approximate the target distribution. In practice, however, the model struggles as complexity increases, performing well on simpler state estimation problems but losing approximation capability as problems become more demanding. The literature on the individual components of the SNF is extensive, yet the literature on model architectures similar to the SNF as a whole is limited. This may indicate that the area is relatively unexplored or that previous attempts have had limited success. Nevertheless, this project sheds light on both the potential and the limitations of this architectural approach, and in doing so, presents a foundation upon which future work could build.

7.3 Future work

The proposed future work focuses on potential solutions to the model's identified challenges, particularly those related to the embedding strategy and data representation. The following topics are believed to improve the model's performance if explored.

- **Different test cases**

  Further investigation into the model's performance is needed. Evaluating the SNF on a wider range of scenarios and gradually increasing problem complexity could yield a better understanding of the conditions under which the embedding model performs poorly.

- **Data representation**

  The difficulty in producing time-independent embeddings is believed to be addressable if the influence of the transition function could be removed from the input data, that is, if the data could be represented in a way that does not reflect the transition dynamics. One candidate representation is the relative residual between a predicted latent state and the corresponding observation, defined as

$$\delta = h(f^k(x_{t-1})) - y_k$$

where k is the number of times the transition function f is recursively applied to the most recent state x_{t-1}.

- **Embedding model**

  Investigating alternative embedding architectures could lead to improved performance. The GRU is a relatively simple model. Transformer-based architectures have demonstrated strong capability in embedding complex sequential structures by learning to weight different parts of the input sequence. In contrast, the GRU tends to place most emphasis on the most recently processed inputs, which may contribute to the observed temporal dependency in the representations.

8

Conclusion

This thesis has investigated how variational inference can be used to approximate the smoothing distribution, which describes the probability distribution of the latent state conditioned on the noisy observations. A model called the Sequential Normalising Flow was presented, consisting of an embedding model and a generative model and evaluated on two Gaussian state-space models, one linear and one nonlinear.

The performance was evaluated based on the model's ability to approximate the true smoothing distribution, using traditional methods such as the Kalman Smoother and the Unscented Kalman Smoother as reference points. The comparison was conducted on both a single observation sequence and multiple sequences, and the computational time of each method was also compared. In the linear case, the SNF yields an approximation that is very close to the Kalman Smoother solution. A small error is observed at the final time steps, which further analysis suggests is a consequence of the embedding model's representational capabilities. For the nonlinear case, the model struggles to obtain a good approximation across the entire sequence, with analysis of the embedding model revealing a similar but more dominant representational issue.

The main conclusion is that the Sequential Normalising Flow shows promising results in the linear case but struggles with the nonlinear case. This difficulty is primarily attributed to the embedding model's inability to construct representations independent of temporal position. It is therefore believed that an embedding which better captures the relative difference between the dynamic prior and the future observations would improve overall performance. The model also shows potential as a computationally efficient alternative for obtaining the smoothing distribution, given further optimisation. More broadly, several avenues remain to be explored, including alternative embedding architectures, reformulating the information provided to the model, and revising the optimisation objective.

In summary, the Sequential Normalising Flow demonstrates promising attributes but has key aspects that need further investigation. Addressing the embedding problem, identified as the critical limitation, could yield a model that is competitive with established methods in both approximation quality and runtime.

Bibliography

[1] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[2] George Cybenko. "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314. DOI: 10.1007/BF02551274.

[3] Laurent Dinh, David Krueger, and Yoshua Bengio. *NICE: Non-linear Independent Components Estimation*. 2015. arXiv: 1410.8516 [cs.LG]. URL: https://arxiv.org/abs/1410.8516.

[4] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. *Density estimation using Real NVP*. 2017. arXiv: 1605.08803 [cs.LG]. URL: https://arxiv.org/abs/1605.08803.

[5] James Durbin and Siem Jan Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, May 2012. ISBN: 9780199641178. DOI: 10.1093/acprof:oso/9780199641178.001.0001. URL: https://doi.org/10.1093/acprof:oso/9780199641178.001.0001.

[6] Mathieu Germain et al. *MADE: Masked Autoencoder for Distribution Estimation*. 2015. arXiv: 1502.03509 [cs.LG]. URL: https://arxiv.org/abs/1502.03509.

[7] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661.

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG]. URL: https://arxiv.org/abs/2006.11239.

[9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[10] Michael I. Jordan et al. "An Introduction to Variational Methods for Graphical Models". In: *Machine Learning* 37.2 (1999), pp. 183–233. DOI: 10.1023/A:1007665907178. URL: https://doi.org/10.1023/A:1007665907178.

[11] Rudolf E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45. DOI: 10.1115/1.3662552.

[12] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[13] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[14] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[15] Review of book by Sebastian Thrun, Wolfram Burgard, and Dieter Fox. “Probabilistic Robotics”. In: *Computer* 39.2 (2006), pp. 95–95. ISSN: 0018-9162. URL: https://research.ebsco.com/linkprocessor/plink?id=b1eeadb7-ee6c-3a24-ba1e-dbc4f11d316f.

[16] Simo Särkkä and Lennart Svensson. *Bayesian Filtering and Smoothing*. 2nd ed. Cambridge University Press, 2023.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY