UPPSALA
UNIVERSITET

Deep Learning for Smoothing in Dynamical Systems

Hannes Larsson, Songchen Li

Project in Computational Science: Report

February 2018

PROJECT REPORT

Institutionen för informationsteknologi

Contents

Abstract

The smoothing problem is a common problem with the goal of finding the probability density function of some hidden state, given some measurements. Different methods using deep learning for solving the Smoothing problem have been implemented. The first two methods are for finding a point estimate for a state space model and is, for the linear and Gaussian case, compared to the mean of the optimal solution provided by the Kalman smoother. Secondly we demonstrate the principle of black box variational inference for a simple problem and for Bayesian smoothing to recover the full posterior probability density function for a state space model.

1 Introduction

Dynamical systems and state space models (SSM) are widely used for systems that evolve in time. Common applications are for instance automatic control systems, positioning systems and weather forecasting. A SSM, sometimes also called hidden Markov model, can be written as

$$\begin{cases} z_t = f(z_{t-1}) + v_{1_{t-1}} \\ x_t = g(z_t) + v_{2_t}, \end{cases} \tag{1}$$

where f and g are functions and v_1 and v_2 are noise terms. Here z_t is called the hidden state and x_t is called the observation. The states at each time step are only influenced by the states in the previous time step, as illustrated in Figure 1. Equation (1) can also be formulated with probability density functions,

$$\begin{cases} Z_t|\{Z_{t-1} = z_{t-1}\} \sim p(z_t|z_{t-1}) \\ X_t|\{Z_t = z_t\} \sim p(x_t|z_t). \end{cases} \tag{2}$$

When put together we can use the properties from the SSM to write the joint probability density function as

$$p(\mathbf{x}, \mathbf{z}) = p(z_0)\Big[\prod_{t=1}^{T} p(z_t|z_{t-1})\Big]\Big[\prod_{t=1}^{T} p(x_t|z_t)\Big], \tag{3}$$

where

$$\mathbf{z} = (z_1, z_2, ..., z_T) \tag{4}$$
$$\mathbf{x} = (x_1, x_2, ..., x_T). \tag{5}$$

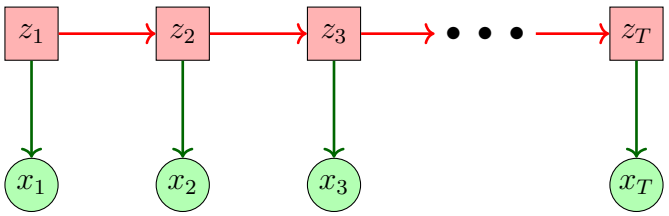


Figure 1: A graphical illustration of a state space model.

Given a state space model it is often of interest to reason about what $\mathbf{z}$ is even though we can only measure $\mathbf{x}$. Finding z_t given some measurements $x_1, x_2, ..., x_t$ is called the filtering problem and finding z_t given all measurements $\mathbf{x}$ is called the smoothing problem.

The main difference between the filtering and the smoothing problem is that in smoothing measurements from the future are taken into account. For filtering only past and current

measurements are used for estimating z_t. The smoothing problem is the main focus of this report. In smoothing, we want to find a posterior PDF

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})} \tag{6}$$

For linear and Gaussian state space models the optimal solutions for filtering and smoothing are called the Kalman filter and Kalman smoother, respectively. The specific Kalman smoother used in this report is the Rauch–Tung–Striebel smoother which we will refer to as the Kalman smoother. While the closed form solutions exist for the linear and Gaussian cases, for non-linear models with non-Gaussian noise the optimal solutions are in general computationally intractable [1]. For instance, to solve Equation (6) one has to calculate $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z}$, this is usually hard to compute.

The aim of this project is to explore different state-of-the-art methods used in solving this problem that also work for non-linear systems with non-Gaussian noise. More specifically, we consider two different existing methods that can be applied to this problem, black box variational inference and convolutional neural networks. We also suggest one method similar to the convolutional neural network that uses a recurrent neural network. The main goal is to verify already existing methods and see that we can recover known distributions using them.

2 Background

2.1 Black box variational inference

The main idea with variational inference is to approximate an unknown probability distribution $p(\mathbf{z}|\mathbf{x})$ with another variational probability distribution $q_\theta(\mathbf{z}|\mathbf{x})$ which is controlled by parameters θ. To approximate $p(\mathbf{z}|\mathbf{x})$, we optimise θ to minimise the Kullback–Leibler (KL) divergence [2, Chapter 5] between $q_\theta(\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$. The KL divergence is a distance used to measure the difference between two distributions, which is defined as

$$\mathrm{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \int q_\theta(\mathbf{z}|\mathbf{x}) \log \frac{q_\theta(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z}. \tag{7}$$

The KL divergence is always positive and the smaller the KL divergence is, the more similar the two distributions are. When the KL divergence is equal to 0, the two distributions will be identical. We can also rewrite the KL divergence as

$$\begin{aligned}\mathrm{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) &= \int q_\theta(\mathbf{z}|\mathbf{x}) \log q_\theta(\mathbf{z}|\mathbf{x}) d\mathbf{z} - \int q_\theta(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}) d\mathbf{z} + \int q_\theta(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}) d\mathbf{z} \\ &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log q_\theta(\mathbf{z}|\mathbf{x})] - \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}, \mathbf{z})] + \log p(\mathbf{x}).\end{aligned} \tag{8}$$

Equation (8) reveals that the KL divergence still depends on $p(\mathbf{x})$, which makes the KL divergence intractable. Since $p(\mathbf{x})$ does not depend on q_θ, minimising the KL divergence

is equivalent to minimising the first two terms $\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log q_\theta(\mathbf{z}|\mathbf{x})] - \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}, \mathbf{z})]$. Therefore, instead of minimising the whole KL divergence, we maximise the Evidence Lower Bound (ELBO) [3], which can be written as

$$\mathcal{L} = \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}, \mathbf{z})] - \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log q_\theta(\mathbf{z}|\mathbf{x})]. \tag{9}$$

Maximising the ELBO is equivalent to minimising the KL divergence, and it allows us to avoid the computation of $p(\mathbf{x})$. However, it is still expensive to calculate the gradient of the ELBO for maximisation, since the gradient of the ELBO consists of the derivatives of expected values with respect to the approximate posterior. This is where black box variational inference comes in. Instead of calculating the expected value with respect to the approximate posterior $q_\theta(\mathbf{z})$, samples $\mathbf{z}$ are drawn from this distribution and stochastic optimization is used to calculate noisy estimates of the ELBO and the gradient of the ELBO [4].

One way of obtaining the noisy gradients of the ELBO

$$\nabla_\theta \mathcal{L} = \nabla_\theta \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}, \mathbf{z})] - \nabla_\theta \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log q_\theta(\mathbf{z}|\mathbf{x})] \tag{10}$$

is to use the reparametrisation trick [5]. The reparametrisation trick is a method used to sample from a distribution parameterised by some parameters θ while separating the stochasticity of the sampling from these parameters. This allows us to compute gradients of expected values as in Equation (10), where the distribution with respect to which the expected value is computed itself depends on the parameters. For instance, if we assume $q_\theta(z_t|\mathbf{x}) = \mathcal{N}(z_t|\mu_\theta, \sigma_\theta^2)$, to implement reparametrisation, we sample ϵ from $\mathcal{N}(0, 1)$, instead of sampling z_t from $\mathcal{N}(\mu_\theta, \sigma_\theta^2)$, then we get z_t by using

$$z_t = \mu_\theta + \sigma_\theta \epsilon =: g_\theta(\epsilon), \tag{11}$$

which is equivalent to sampling z_t from $\mathcal{N}(\mu_\theta, \sigma_\theta)$. After using the reparametrisation trick, we can compute the gradient of z_t with respect to μ_θ and σ_θ, then we can obtain the gradient with respect to θ using the chain rule. Specifically, the expectation terms with respect to $q_\theta(z_t|\mathbf{x})$ can be rewritten as

$$\nabla_\theta \mathbb{E}_{q_\theta(z_t|\mathbf{x})}[f(\mathbf{x}, z_t)] = \nabla_\theta \mathbb{E}_{p(\epsilon)}[f(\mathbf{x}, g_\theta(\epsilon))] = \mathbb{E}_{p(\epsilon)}[\nabla_\theta f(\mathbf{x}, g_\theta(\epsilon))] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta f(\mathbf{x}, g_\theta(\epsilon_i)). \tag{12}$$

In Equation (12), N is the number of samples we used to approximate the expected value and in general f can be any function as long as it takes z_t as input.

This reparametrisation trick will be used throughout this report.

2.2 Neural networks

Neural networks (NN) are functions consisting of multiple functions, or neurons. The neurons are represented by a composition of multiple different functions. For instance a

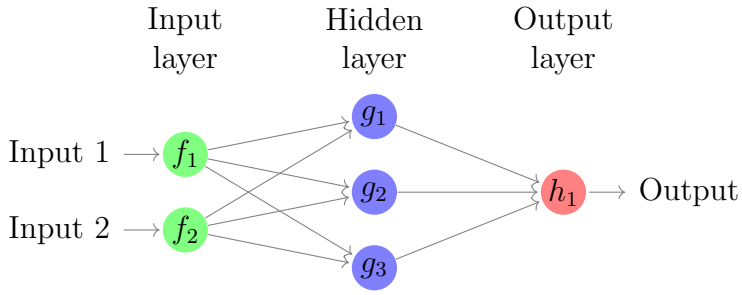


Figure 2: A graph for a basic neural network which consists of an input layer, a hidden layer and an output layer.

function $g(\mathbf{x})$ can be composed of other functions $f(\mathbf{x})$ who are in turn composed of other functions and so on. This can be graphically illustrated with a network with neurons as nodes and arrows showing the dependencies between the nodes. An illustration of a NN with two input neurons, a single hidden layer with three neurons and a single output neuron can be seen in Figure 2. Typically the neuron function is

$$g_i(\mathbf{x}) = K(\sum_t \omega_t f_t(\mathbf{x}) + b), \tag{13}$$

where ω_i are called weights and b is a bias. K is some non linear function, called the activation function. Usually, the network is trained by optimising a cost function with respect to the weights and biases. [2, Chapter 6]

One common way to optimise a NN is by using stochastic gradient descent (SGD). Usually we have some loss function that we want to minimise for each training sample. For a large set of training samples, the gradient becomes computationally hard to compute. Instead of computing the gradient based on all training samples, we can use the fact that the gradient of all training samples is an expected value that we can approximate by taking the average of the loss function over a small sample set, called a minibatch. To compute the gradient, an algorithm called backpropagation is used, which calculates the gradient layer by layer backward from the loss function to the input layer. [2, Chapter 5]

2.2.1 Convolutional neural networks

Convolutional neural networks (CNN) are a type of NNs that are commonly used in image recognition. Instead of using a general linear combination of the previous neurons as described in Equation (13) it uses cross correlation. Cross correlation is similar to convolution to the input data with a number of different convolutional kernels. The convolutional kernel is moved across the entire input signal and computing the dot product of the kernel and the different regions in the input. Then the output is passed through a nonlinear activation function. The output is a new signal similar to the input signal for each different convolution kernel. Then the output is a new signal which can be used as the input of next layer. [2, Chapter 9]

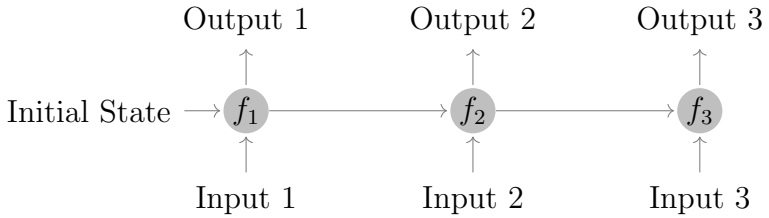


Figure 3: Graph for a three layers recurrent neural network. The computation in each neuron not only depends on the input, but also depends on the state/output of the previous neuron. For the first neuron, its computation depends on the pre-set initial state and the first input, since it does not have a previous neuron. Because the state and the output are identical in this RNN, the computation result of each neuron can be used as both output of the current state and the previous state for next neuron.

2.2.2 Recurrent neural networks

Recurrent neural networks (RNN) are a type of NNs that can use sequential information to compute the output of current state. The sequential information can be acquired from past, future, or both. Thus, RNNs are especially suitable for learning time sequence data. In the RNN we used in this report, the output of each state was also fed into the next RNN cell. This is illustrated in Figure 3.

3 Delimitations and Numerical Implementation

As mentioned in the introduction, the main goal of this project was to verify already existing methods and see that if we could recover known distributions using them. Then for some of the methods we ran them on models with unknown posterior distribution for demonstration. Python 3 was used to implement all the methods mentioned in this report. Furthermore, we have used the library 'TensorFlow' to implement all the neural networks mentioned in this report. We also used the 'CUDA® Toolkit 8.0' and 'cuDNN v6.1' to run 'TensorFlow' on a GPU card to accelerate the training of the CNN. Considering the limitation of time and hardware, all implementations here were done for one-dimensional states and observations, i.e. x_t and z_t will always be scalars for all t. Performance related issues like how much time it takes to train the different methods etc. are beyond the scope of this report and not considered here.

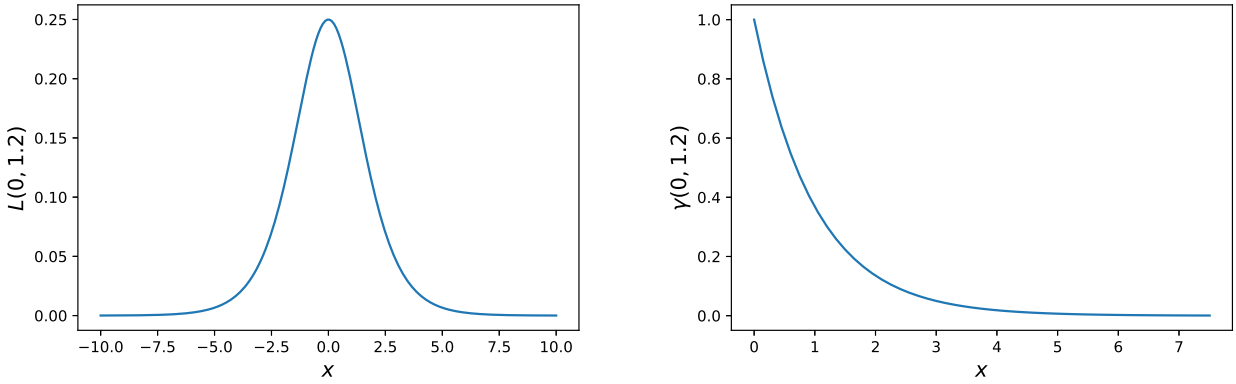
4 Generative models

The generative models used in the experiments were all on the form of Equation 1, with functions f, g and noise terms v_{1_t}, v_{2_t} as described in Table 1. We have used four different generative models, Linear Gaussian (LG), Non-linear Gaussian (NLG), Linear non-Gaussian (LNG) and Non-linear non-Gaussian (NLNG). For all models $z_0 = 0$. The functions α and β in Table 1 are defined as follows:

$$\alpha(z) = \begin{cases} 0.9\tanh(z), & z < 1 \\ 1 - 0.9\tanh(z), & z \geq 1 \end{cases}$$

Table 1: Function parameters for the different generative models used in this report.

	$f(z)$	$g(z)$	v_1	v_2
LG	$0.9z$	$3.5z$	$\mathcal{N}(0,1)$	$\mathcal{N}(0,1)$
NLG	$\alpha(z)$	$\beta(z)$	$\mathcal{N}(0,1)$	$\mathcal{N}(0,1)$
LNG	$0.9z$	$3.5z$	$L(0,1.2)$	$\gamma(1)$
NLNG	$\alpha(z)$	$\beta(z)$	$L(0,1.2)$	$\gamma(1)$



(a) Logistic distribution $L(0, 1.2)$

(b) Gamma distribution $\gamma(1)$

Figure 4: Plots of the $L(0, 1.2)$ and $\gamma(1)$ as used in the LNG and NLNG models.

and

$$\beta(z) = z^3$$

for the NLG and NLNG models. The noise terms presented in Table 1 that are not Gaussian noise are probability density functions which the noise is drawn from,

$$L(a, b) = \frac{e^{-\frac{x-a}{b}}}{b(1 + e^{-\frac{x-1}{b}})^2}$$

and

$$\gamma(a) = x^{a-1}\frac{e^{-x}}{\Gamma(a)},$$

where Γ is the gamma function [6]. $L(a, b)$ is called the Logistic distribution and $\gamma(a)$ is the Gamma distribution. These PDFs for the values of a and b described in Table 1 can be seen in Figure 4.

5 Point Estimation Smoothers

Two different methods for estimating point values of the hidden variables $\mathbf{z}$ were investigated during this project. One based on a convolutional neural network and one based on a recurrent neural network. For training we used 20 000 different time series and for the test data we used 5 000 time series. These two methods are useful when we have a good generative model to simulate both measurements and hidden states.

5.1 ConvNet Smoother

The ConvNet smoother is a method that uses a CNN with dilated convolutional layers to find a mapping from the input $\mathbf{x} = (x_1, x_2, ..., x_T)$ to and output $\hat{\mathbf{z}} = (\hat{z}_1, \hat{z}_2, ..., \hat{z}_T)$ that minimises the pseudo-Huber loss function [7],

$$L = \sum_{t=1}^{T} \left(\sqrt{1 + (z_t - \hat{z}_t)^2} - 1 \right). \tag{14}$$

Here z_t is the tth term of $\mathbf{z}$.

A dilated convolutional layer is like a regular convolutional layer except the elements of the kernel have holes between them. The size of the hole is determined by a dilation factor. A dilated convolutional kernel of length a with a dilation factor b is equivalent to a regular convolutional kernel which has $b-1$ zeros between each elements in the dilated kernel and whose length is $a+(a-1)(b-1)$. A regular convolutional layer can be seen as a dilated convolutional layer with dilation factor one. We used the same hyperparameters as in [8]. Data was generated with $T = 200$ and we used seven dilated convolutional layers with 60 different convolutional kernels in each layer. The dilation factor was one in the first two layers and then doubled for the following layers. After the seven dilated convolutional layers, one fully connected layer was made, where the outputs of the last dilated convolutional layer were concatenated and put into a single layer in which the neurons are described by Equation (13) for each $t \in [1, ..., T]$.

5.2 RNN for a point estimate

The RNN for a point estimate is a method inspired by the methods used in a paper of BBVI for the smoothing problem [9] and the ConvNet smoother [8]. Figure 5 shows the structure of the neural network.

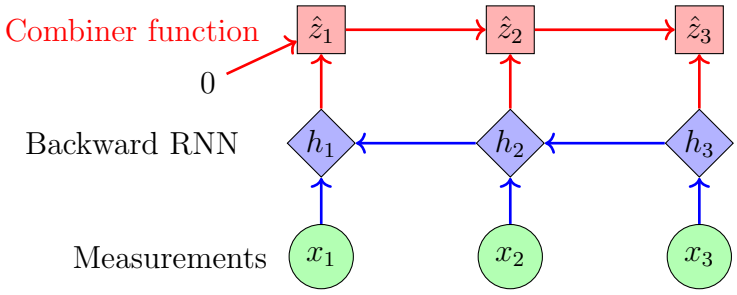


Figure 5: RNN for a point estimate

The whole network consists of one RNN that goes backward in time and in which we define a new series of hidden states $h_1, h_2, ..., h_T$. What this means is that we determine the hidden state h_t as a non-linear function of the current measurement x_t and the next hidden state h_{t+1}. More specifically, h_t can be written as

$$h_t = \text{ReLU}(w_x x_t + w_h h_{t+1} + b). \tag{15}$$

For the last time step, we set $h_{T+1} = 0$ to compute h_T. After collecting the information from current and future measurements to h_t, we used another non-linear function h_t^c to take the information from the last state $\hat{z}_{t-1}$ to compute the current state $\hat{z}_t$. Therefore, for h_t^c and z_t we have

$$h_t^c = \frac{1}{2}(\tanh(w_c \hat{z}_{t-1} + b_c) + h_t) \tag{16}$$

$$\hat{z}_t = w_z h_t^c + b_z \tag{17}$$

for $t = 1, 2, 3, ..., T$. We manually set $\hat{z}_0 = 0$ as the initial state. We used $\hat{z}_1, \hat{z}_2, ..., \hat{z}_T$ as the outputs of the RNN, then for training we minimized the sum of the mean squared error:

$$\text{Loss} = \sum_j \frac{1}{T} \sum_{t=1}^{T} (z_{j_t} - \mu_{jt})^2. \tag{18}$$

Here j is the index for the different samples in a batch. We used gradient descent to optimise the loss function.

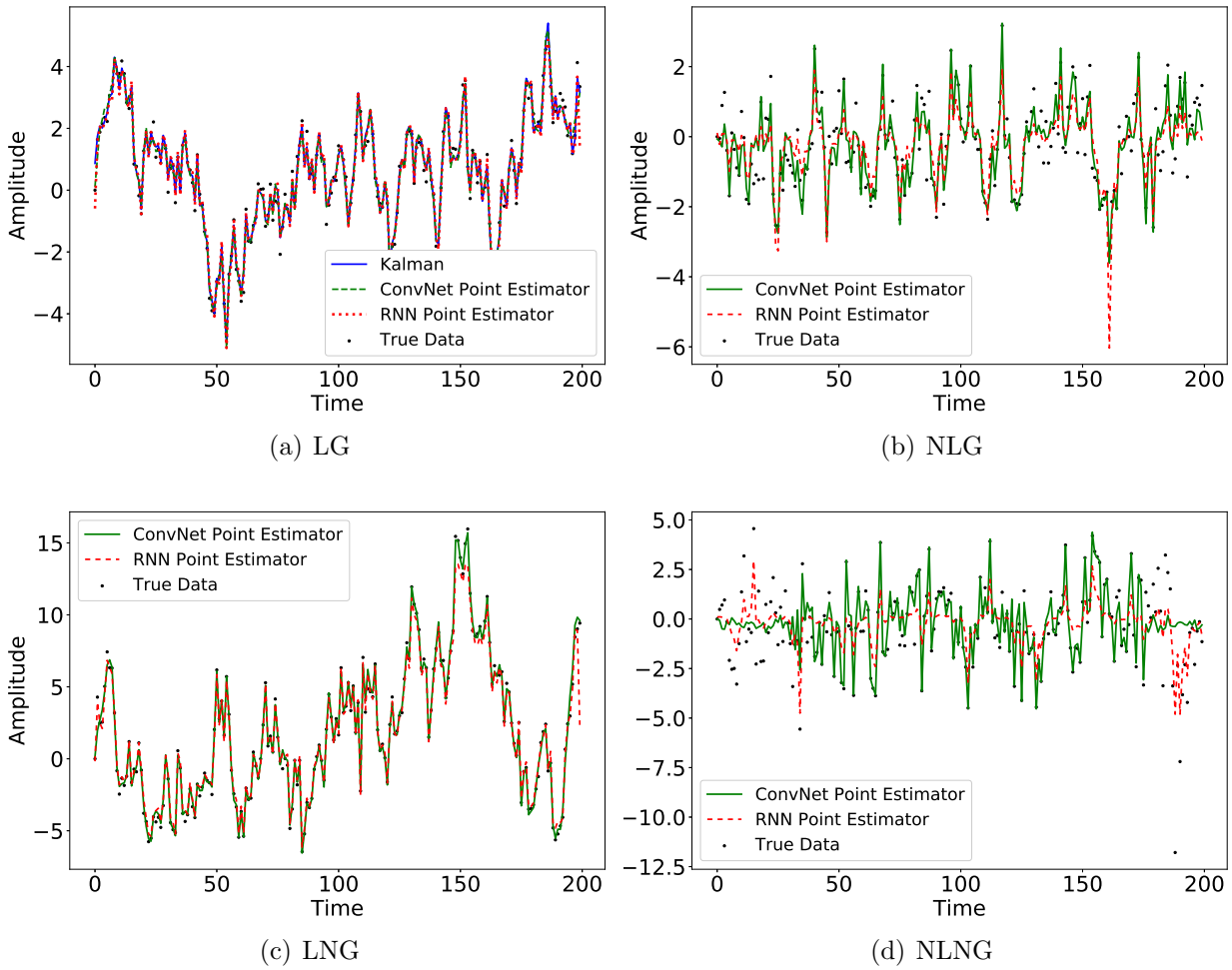
5.3 Results

In Table 2 the root mean squared error (RMSE) for the ConvNet smoother and the RNN point estimator is shown. The error of a prediction of $z_t = 0 \forall t$ is also shown in the table. For the LG model the RMSE for the Kalman filter is shown as a ground truth comparison.

Table 2: Root mean squared error for the Kalman filter, ConvNet smoother, RNN point estimator and the estimate that z = 0 for all time steps.

RMSE	LG	NLG	LNG	NLNG
Kalman	0.270			
ConvNet	0.34	0.53	0.40	1.24
RNN point est.	0.332	0.693	0.535	1.53
$z \equiv 0$	2.265	1.123	4.928	2.247

In Figure 6 examples of outputs from the ConvNet smoother and the RNN point smoother can be seen. Looking at Table 2 and Figure 6 it is clear that these models work well for LG and LNG. It is harder to judge the performance for the NLG and NLNG data sets. They perform better than the constant prediction $z = 0$ but if this can be useful in practice or not is determined by the application. For the NLNG data set especially the RNN point estimator looks quite bad, but it is hard to make a proper judgement with no ground truth to use for comparison.



(a) LG

(b) NLG

(c) LNG

(d) NLNG

Figure 6: Plots of the point estimation smoothers. The ConvNet smoother, RNN point estimator and the true values of z are shown. For the LG data, the Kalman smoothed signal is also shown as a ground truth for comparison.

5.4 Improvements

For the RNN point estimator we used the MSE as the loss function. However, as mentioned in [8], the pseudo-Huber loss function [7] has better computational properties. So, changing the loss function for RNN point estimator to the pseudo-Huber loss function is likely to give us a better result.

One more limitation of our presented results is the fact that we do not have a good way of verifying the validity of the point estimates where there does not exist a ground truth. This was handled by comparing the RMSE to a prediction that does not take the observations into account at all, namely that $z \equiv 0$. We could verify that our method provided better predictions than this but it would be good to be able to draw a more trustworthy conclusion.

6 Variational Inference

6.1 Black-box Variational Inference for a static 1D problem

Before using BBVI for smoothing we consider a simple static and one-dimensional problem to illustrate the basic BBVI method. This implementation is similar to what is described in the paper from 2013 [4], but with a different expression for the ELBO. We also used the reparametrisation trick to control the variance instead of Rao-Blackwellization as described in [4]. The model used was:

$$\begin{cases} p(z) = \mathcal{N}(z|\mu_z, \sigma_z^2) \\ p(x|z) = \mathcal{N}(x|Az, \sigma_x^2) \\ p(x,z) = p(z)p(x|z). \end{cases} \tag{19}$$

We used one measurement x sampled from $p(x|z)$ and used BBVI to find $p(z|x)$. For a model like the one described in Equation (19), there are analytical expressions for the posterior probability density function [10], which could provide a ground truth to compare the optimised approximate posterior with.

Now recall the ELBO, described in Equation (9),

$$\mathcal{L} = \mathbb{E}_{q_\theta(z|x)}[\log p(x,z)] - \mathbb{E}_{q_\theta(z|x)}[\log q_\theta(z|x)]. \tag{20}$$

If the approximate posterior q_θ is single variable Gaussian distributed, the term $-\mathbb{E}_{q_\theta(z|x)}[\log q_\theta(z|x)]$ has a closed analytical form

$$-\mathbb{E}_{q_\theta(z|x)}[\log q_\theta(z|x)] = \frac{1}{2}(1 + \log(2\pi)) - \frac{1}{2}\log\sigma_\theta^2. \tag{21}$$

A detailed derivation of Equation (21) for a Gaussian approximate posterior q_θ can be seen in appendix A.

We used noisy gradients to estimate the gradient of the ELBO instead of using the exact gradient, which is the reason why this method is called Black-box Variational Inference. We sampled z from $q_\theta(z)$ and plugged that into the expression for the gradient of the ELBO. More specifically we used the reparametrisation trick, introduced in Section 2.1, for the calculation of the gradients. That is, we sampled $\epsilon \sim \mathcal{N}(0,1)$ which makes

$$z = \mu_\theta + \sigma_\theta \epsilon \tag{22}$$

a sample from $q_\theta(z) = \mathcal{N}(\mu_\theta, \sigma_\theta^2)$. Then we can calculate the gradient of the ELBO given in Equation (20) with respect to μ_θ and σ_θ and evaluate that expression as the mean of a large number (we used 100) of sampled z for each optimisation step.

The optimisation steps were chosen as

$$(\mu_{\theta_{t+1}}, l_{\theta_{t+1}}) = \mathbf{v}_{t+1} = \mathbf{v}_t + \alpha_t \nabla \mathbf{v}_t, \tag{23}$$

where $l = \log \sigma$ and α_t is a step size which follows the Robbins-Monro conditions [4]. l was chosen instead of σ to to provide stability for the algorithm. The standard deviation is strictly positive and the transformation of σ makes the optimisation problem unconstrained by transforming the feasible set of the optimisation from $\mathbb{R} \times \mathbb{R}^+$ to $\mathbb{R}^2$.

6.1.1 Results

The ELBO described in the previous section was maximised for the model in Equation. 19, with parameters $\mu_z = 4$, $\sigma_z^2 = 1$, $A = 5$ and $\sigma_x = 1$. The initial guess was

$$\mu_\theta = 30$$
$$\sigma^2 = 10.$$

In Figure 7, a plot of the trace of the optimisation algorithm can be seen, along with level curves for the ELBO and the analytically computed posterior. We can see that the optimisation algorithm does indeed converge to the analytically calculated posterior. One can also see that the curve is not perpendicular to the objective function which makes sense since the optimisation is stochastic, thus the stochastic gradients can differ from the true gradients.

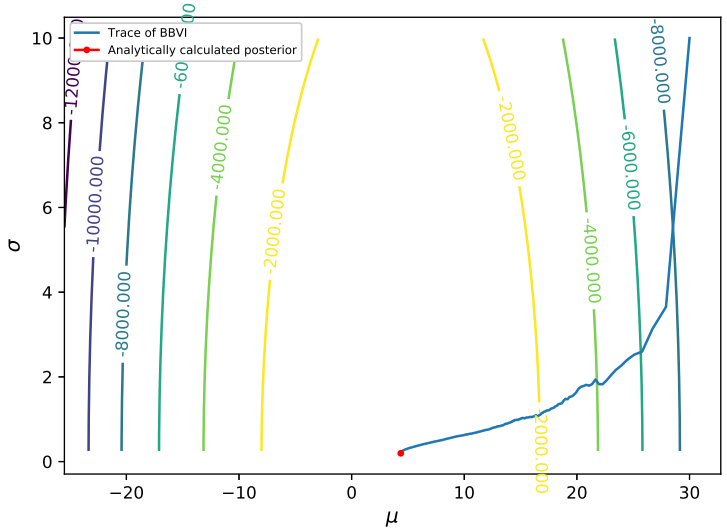


Figure 7: The trace of the optimisation with initial guess $(\mu, \sigma) = (30, 10)$ that converges to the analytical posterior PDF.

6.2 Structured Variational Inference

The structured variational inference is a method of variational inference using noisy gradients introduced in a recent paper [9]. We have reproduced a part of the method here. In the original paper, they use data to both learn a SSM and do the inference. The purpose of our method is to show that we can use this method to get the same posterior as in a Kalman filter. We have a known SSM and use a similar network as in Section 5.2

to parametrise the expected value μ_t and the standard deviation σ_t of the approximate posterior $q_\theta(\mathbf{x})$. μ_t and σ_t are functions of $h_{combined}$ as described in Equation 16,

$$\mu_t = W_\mu h_{combined_t} + b_\mu, \tag{24}$$
$$\sigma_t = \text{softplus}(W_\sigma h_{combined} + b_\sigma). \tag{25}$$

The main difference from the RNN for point estimate is that the objective function is the ELBO instead of the sum of mean squared errors. The gradients of the ELBO are calculated using the reparametrisation trick and backpropagation for a large number of samples. Using properties of the SSM we can assume that the approximate posterior only depends on the previous value of the hidden variable and the current and future measurements. With this in mind, we factorise the approximate posterior

$$q_\theta(\mathbf{z}|\mathbf{x}) = q_\theta(z_1|x_1, ..., x_T) \prod_{t=2}^{T} q_\theta(z_t|z_{t-1}, x_t, ..., x_T). \tag{26}$$

Then we make the ansatz that

$$q_\theta(z_t|z_{t-1}, x_t, ..., x_T) \sim \mathcal{N}(\mu_t(z_{t-1}, x_t, ..., x_T), \sigma_t^2(z_{t-1}, x_t, ..., x_T)). \tag{27}$$

The ELBO in this case is

$$\mathcal{L} = \sum_{t=1}^{T} \mathbb{E}_{q_\theta(z_t)}[\log p(x_t|z_t)] - \text{KL}(q_\theta(z_1)||p(z_1)) - \sum_{t=2}^{T-1} \mathbb{E}_{q_\theta(z_{t-1})}[\text{KL}(q_\theta(z_t|z_{t-1})||p(z_t|z_{t-1}))], \tag{28}$$

according to [9]. A detailed derivation of this can be seen in the appendix of [9]. One way of rewriting this to make the expression compatible with Equation (12) is

$$\mathcal{L} = \sum_{t=1}^{T} \mathbb{E}_{p(\epsilon_t)}[f_{\theta_t}(x_t, g_{\theta_t}(\epsilon_t))] - \text{KL}(q_\theta(z_1)||p(z_1)), \tag{29}$$

where

$$f_{\theta_t}(x, g_{\theta_t}(\epsilon_t)) = \log p(x_t|g_{\theta_t}(\epsilon_t)) - \text{KL}\Big(q_{\theta_t}(z_{t+1}|g_{\theta_t}(\epsilon_t))||p(z_{t+1}|g_{\theta_t}(\epsilon_t))\Big), \tag{30}$$

for $t = 1, ..., T-1$ and

$$f_{\theta_T}(x, g_{\theta_T}(\epsilon_T)) = \log p(x_T|g_{\theta_T}(\epsilon_T)) \tag{31}$$

for T. Then we can proceed and approximate

$$\nabla_\theta \mathbb{E}_{p(\epsilon_t)}[f_{\theta_t}(x_t, g_{\theta_t}(\epsilon_t))] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta f_{\theta_t}(x_t, g_{\theta_t}(\epsilon_t^i)). \tag{32}$$

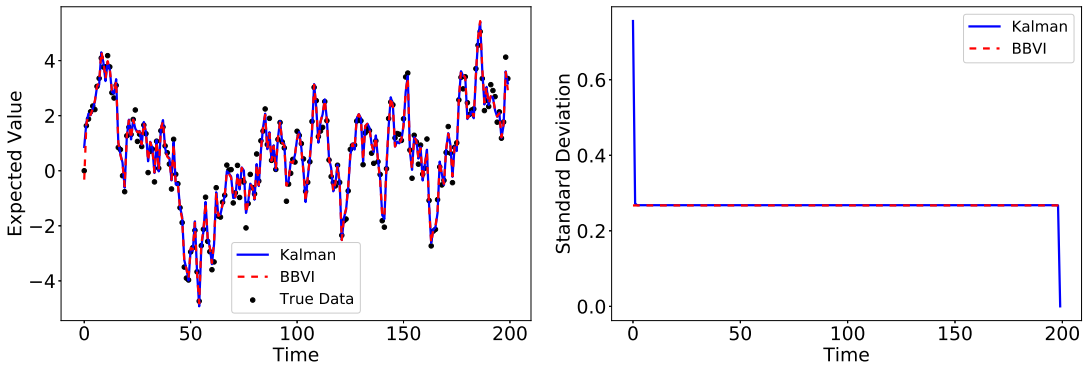
Then we sampled ϵ_t^i, $i = 1, .., N$and $t = 1, ..., T$, and maximised the function

$$\text{L} = \frac{1}{N} \sum_{i=1}^{N} \big[\sum_{t=1}^{T} f_{\theta_t}(x, g_{\theta_t}(\epsilon_t^i)) - \text{KL}(q_\theta(z_1)||p(z_1)) \big] \tag{33}$$

using the Adam optimiser [2, Chapter 8] for $\mathbf{x}$ as a single time series generated by the generative model LG with $T = 200$ time steps.

6.2.1 Results

Here the results are presented for two approaches. First, Figure 8 shows the result of the optimisation of the ELBO for a single LG time series along with the Kalman filter and the true data points z. We used a batch size $N = 10$ for each optimisation step. The plot shows the mean and the standard deviation for the result of the structured inference in two separate plots. It is clear that the optimised output manages to recreate the posterior PDF of the Kalman smoother.



(a) Expected Value (b) Standard Deviation

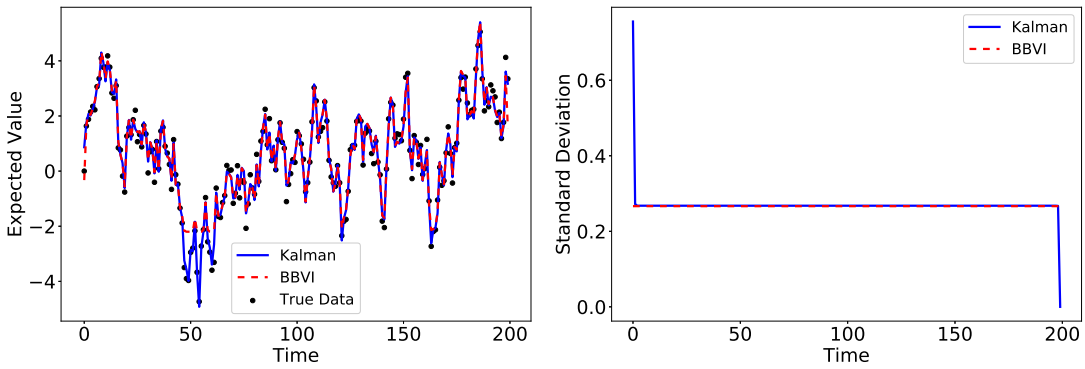
Figure 8: The results of BBVI trained on one sample. After training to converge, BBVI is capable of recovering nearly the same mean and the standard deviation as the Kalman smoother.

Secondly, we trained a RNN for 20 different time series to maximise the ELBO and then verified the predictions on test data compared to the Kalman filter. The number of samples from the posterior PDF in each time was $N = 10$ in this approach aswell. The number of time series considered for each training step was one due to limited memory in the computer the training was done on. Due to limited computational resources this neural network was not trained optimally. Figure 9 shows the output of a typical test data sample of the neural network, which shows some promise for a fully trained NN of this kind.

7 Comparison between the different approaches

The two concepts explored in this report have some differences. Both point estimation methods do not require knowledge of the SSM and learn the estimate from a large batch of artificially generated data. This data can only be generated when the model already exists since the loss function uses the hidden variables that are not observable in reality. Thus, these methods work when a good model already exists. Then you can train a network and use it for point estimation smoothing with real world data as input.

The black box variational inference method implemented here uses the SSM in the expression for the ELBO. Furthermore this method provides a Bayesian solution that gives



(a) Expected Value (b) Standard Deviation

Figure 9: The result of BBVI trained on 20 samples and tested on one test sample. Due to limited computational resource, we could not train the RNN until full convergence.

the full posterior PDF and not only a point estimate. This can be very useful since we can quantify the uncertainties. Hence, for our implementation, knowledge of the underlying model is required for the structured inference method presented here. However, the model does not need a large set of generated data for training or the hidden variables for the loss function, given that the underlying model is known. As demonstrated here, we can find the posterior PDF for a single time series by just maximising the ELBO, that is, we don't need any more data than the actual measurements to get the posterior PDF. We have also shown that training on a small set of time series can make the RNN predict the posterior PDF of other time series with the same underlying model, even though the RNN was not properly trained due to limited computational resources. In the paper by Krishnan et. al. [9] they also learn the SSM by training on a larger data set, i.e. this does not require knowledge of the model beforehand. The form of the posterior is however assumed beforehand, like for instance that the posterior is Gaussian distributed.

8 Discussion

For both point estimation methods, the structures of the RNN and the ConvNet can be further optimised. The loss function of the RNN point estimator could be changed to the pseudo-Huber loss function [7] from the mean squared error for better numerical properties. The parameters in the update algorithm can also be adjusted for fast training speed and better convergence. Dealing with the problem of not having any ground truth to compare our smoothed signals for the NLG, LNG and NLNG models is also worth investigating.

For BBVI, the most interesting future work would be to parameterise the generative model, then train the generative model and the RNN at the same time so that we can implement BBVI without the knowledge of the generative model like it was done by Krishnan et. al. [9]. For the trained RNN, training on a larger batch of data for a longer time can also be an improvement. Finally rewriting the ELBO for some models that are not linear with Gaussian white noise to verify the BBVI method is a simple way to make

the work implemented here more general and useful.

9 Acknowledgements

We would like to thank our supervisor, Fredrik Lindsten for all the great help and guidance we have gotten.

References

[1] Simo Särkkä. *Bayesian Filtering and Smoothing*. Camebridge University Press, chapter 1, 2013.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[4] Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. In Samuel Kaski and Jukka Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 814–822, Reykjavik, Iceland, 22–25 Apr 2014. PMLR.

[5] Archer Evan, Park Il Memming, Buesing Lars, Cunningham John, and Paninski Liam. Black box variational inference for state space models. *arXiv preprint arXiv:1511.07367*, 2016.

[6] Philip J Davis. Leonhard euler's integral: An historical profile of the gamma function. *The American Mathematical Monthly, vol. 66 (1959), pp. 849-869*, 1959.

[7] G. Aubert P. Charbonnier, L. Blanc-Feraud and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on Image Processing, vol. 6, no. 2, pp. 298–311*, 1997.

[8] Ambrogioni Luca, Guclu Umut, Maris Eric, and van Gerven Marcel A. J. Estimating nonlinear dynamics with the convnet smoother. *arXiv preprint arXiv:1702.05243*, 2017.

[9] Krishnan Rahul G., Shalit Uri, and David Sontag. Structured inference networks for nonlinear state space models. *arXiv preprint arXiv:1609.09869v2*, 2016.

[10] Lindsten Fredrik, Schön Thomas B., Svensson Andreas, and Wahlström Niklas. Probabilistic modeling - linear regression & gaussian processes. Course compendium, 2017.

A Derivation of equation 21

Equation 21 for $p(z|x) \sim \mathcal{N}(\mu, \sigma^2)$, is given by

$$
\begin{aligned}
&- \mathbb{E}_{q_\theta(z|x)}[\log q(z|x)] \\
&= -\mathbb{E}_{q_\theta(z|x)}\big[\log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2}\frac{(z-\mu)^2}{\sigma^2}\Big)\big] \\
&= -\mathbb{E}_{q_\theta(z|x)}\big[-\frac{1}{2}\log\big(2\pi\sigma^2\big) - \frac{1}{2}\frac{(z-\mu)^2}{\sigma^2})\big] \\
&= \frac{1}{2}\log\big(2\pi\sigma^2\big) + \frac{1}{2}E_{q_\theta(z|x)}\big[\frac{(z-\mu)^2}{\sigma^2}\big] \\
&= \frac{1}{2}\log(2\pi) + \frac{1}{2}\log\sigma^2 + \frac{1}{2}\frac{\mathbb{E}_{q_\theta(z|x)}(z-\mu)^2}{\sigma^2} \\
&= \frac{1}{2}\log(2\pi) + \frac{1}{2}\log\sigma^2 + \frac{1}{2}\frac{\sigma^2}{\sigma^2} \\
&= \frac{1}{2}(1+\log(2\pi)) + \frac{1}{2}\log\big(\sigma^2\big).
\end{aligned}
$$