EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät

Methoden des Maschinellen Lernens

Masterarbeit

Composing Neural Networks and Probabilistic Graphical Models

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Methoden des Maschinellen Lernens
Johannes van den Heuvel, johanheuvel5@gmail.com, 2023

Bearbeitungszeitraum: August2022-February2023

Betreuer/Gutachter: Prof. Dr. Philipp Hennig, Universität Tübingen
Zweitgutachter: Prof. Dr. Jakob Macke, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Johannes van den Heuvel (Matrikelnummer 5714872), February 13, 2023

Acknowledgments

I want to thank Marius Hobbhahn for providing practical supervision and the weekly meetings.

Contents

1. Introduction

Neural networks are popular Machine Learning models, and deep learning is currently the main paradigm. One of the reasons for this popularity is their flexibility. However, this flexibility comes with a downside and neural networks are also "black-boxes". While this is not a crucial issue in predictive or generative performance it is in other applications.

One such application is scientific modeling. Neural networks are not well suited for this task, as they are difficult to interpret. Even if neural networks can model something and perform well on a task, the model is often not understandable for humans. However, even within predictive or generative tasks black-box models have downsides. For example, modularity is a key principle in software development, and black-box models are not modular. This makes it more difficult to debug and test models.

In this thesis, we will take a look at a model that combines deep learning with another paradigm, probabilistic machine learning. The *Structured Variational Autoencoder*[1], is a composition between a neural network and a probabilistic graphical model. Because of this composition, the model is both flexible enough for complex data and also interpretable.

The contribution of this thesis is twofold. First, reducing the technical debt of the original work [JDW+16]. This thesis provides a more detailed explanation of the background, and mathematical details required to understand it. Furthermore, the original code is not up to date and does not support easy replication of the experiments. This thesis provides a new implementation of the model and experiments using PyTorch. Second, this thesis investigates some of the strengths and weaknesses of the composition. In general probabilistic graphical models are more interpretable than neural networks, which also holds for the composition. Because we can interpret the model better, it is also easier to tweak specific parts after training.

[1] previously introduced in [JDW+16].

2. Background

This chapter contains the mathematical background needed to understand the Structured Variational Autoencoder algorithm. First, we will look at *Expectation Maximization* (section 2.1) which will be used to understand the basics of probabilistic inference. Expectation maximization is an iterative method to find point estimates for parameters. Rather than finding point estimates, we could also use *Variational Inference* (section 2.2) to fit an entire posterior distribution. It is possible to "automate" variational inference using *Variational Message Passing* (section 2.3). Finally, the *Variational Autoencoder* (section 2.4) is a way to scale up variational inference.

2.1. Expectation Maximization (EM)

Expectation Maximization (EM) is a general method to maximize the likelihood for probabilistic models having latent variables [DLR77, MK07]. EM provides the basis for later discussion of Variational Inference.

2.1.1. Evidence Lower Bound (ELBO)

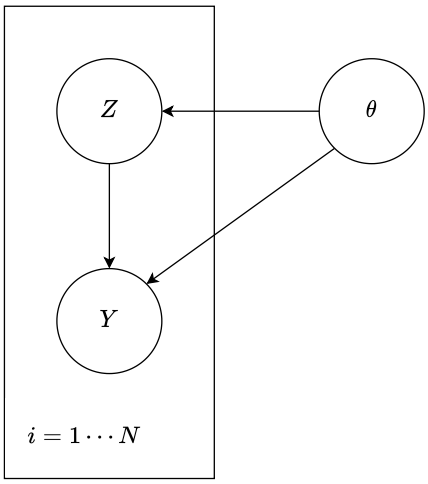


Figure 2.1.: Graphical representation of a generative model.

Consider a probabilistic model with observed random variables Y, latent random variables Z, and parameters θ, as in Figure 2.1. Note that θ is a collection of

parameters, i.e. the prior and likelihood depend on different subsets of θ. We have the following generative process for sample i:

1. sample latent $Z^{(i)}$ from the prior distribution $P(Z|\theta^*)$
2. sample observation $Y^{(i)}$ from the likelihood $P(Y|Z,\theta^*)$

Then we want to infer what the most likely parameters θ are that produced the observed $Y^{(i)}$'s. We want to find θ such that the likelihood $P(Y|\theta)$ is maximized:

$$\theta^* = \max_\theta P(Y|\theta) = \max_\theta \sum_Z P(Z,Y|\theta), \tag{2.1}$$

where the second equality follows from marginalization.

Suppose it is not feasible to directly optimize $P(Y|\theta)$, and that optimization of $P(Z,Y|\theta)$ is much easier. Applying the chain rule to the right side term in (2.1) results in the following [1]

$$P(Z,Y) = P(Z|Y)P(Y) \tag{2.2}$$

$$\Leftrightarrow P(Y) = \frac{P(Z,Y)}{P(Z|Y)} \tag{2.3}$$

Applying the logarithm and expectation $\mathbb{E}_{Q(Z)}$, which depends on a new distribution $Q(Z|\theta)$ also parameterized by θ, to both sides:

$$\begin{aligned} \mathop{\mathbb{E}}_{Q(Z)}\Big[\log P(Y)\Big] &= \mathop{\mathbb{E}}_{Q(Z)}\left[\log \frac{P(Z,Y)}{P(Z|Y)}\right] \\ &= \mathop{\mathbb{E}}_{Q(Z)}\Big[\log P(Z,Y)\Big] - \mathop{\mathbb{E}}_{Q(Z)}\Big[\log P(Z|Y)\Big]. \end{aligned} \tag{2.4}$$

The left-hand side does not depend on $Q(Z)$, so the expectation can be dropped. Adding and subtracting $\mathbb{E}_{Q(Z)}\Big[\log Q(Z)\Big]$ gives

$$\begin{aligned} \log P(Y) &= \mathop{\mathbb{E}}_{Q(Z)}\left[\log P(Z,Y) - \log Q(Z)\right] \\ &\quad - \mathop{\mathbb{E}}_{Q(Z)}\left[\log P(Z|Y) - \log Q(Z)\right] \\ &= \mathop{\mathbb{E}}_{Q(Z)}\left[\log \frac{P(Z,Y)}{Q(Z)}\right] - \mathop{\mathbb{E}}_{Q(Z)}\left[\log \frac{P(Z|Y)}{Q(Z)}\right] \\ &= \mathcal{L}(Q) + \mathbb{KL}(Q||P) \\ &\geq \mathcal{L}(Q). \end{aligned} \tag{2.5}$$

Where the inequality follows from the $\mathbb{KL}(Q||P)$ always being positive. The function $\mathcal{L}$ is called the *Evidence Lower Bound* (ELBO), as it lower bounds the evidence term on the left. And the final result is

$$\log P(Y) \geq \mathcal{L}(Q) \tag{2.6}$$

[1] θ is omitted for notational convenience.

2.1.2. EM algorithm

The Expectation Maximization (EM) algorithm consists of two steps, the expectation step and maximation step, between which we alternate: [2]

$$\text{E-step:} \quad \max_{Q} \mathcal{L}(Q;\theta), \tag{2.7}$$

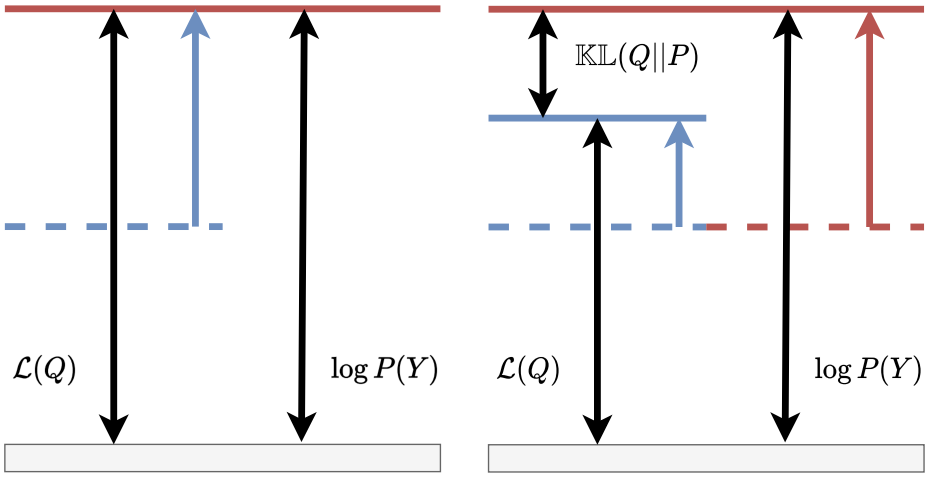
$$\text{M-step:} \quad \max_{\theta} \mathcal{L}(Q;\theta). \tag{2.8}$$

In the E-step (2.7) $\mathcal{L}$ is maximized w.r.t. the distribution Q. Recall the definition of the ELBO

$$\mathcal{L}(Q) = \log P(Y) - \mathbb{KL}(Q||P) \tag{2.9}$$

Optimizing the ELBO w.r.t. Q is equivalent to maximizing $-\mathbb{KL}(Q||P)$ as $P(Y)$ is constant w.r.t. $Q(Z)$. As the $\mathbb{KL}$ is always positive it follows that this maximized when $-\mathbb{KL} = 0$ and thus $Q^* = P$.

In the M-step (2.8) we maximize $\mathcal{L}$ w.r.t. the parameters θ. This can be achieved by both increasing $\log P(Y)$ and decreasing $-\mathbb{KL}(Q||P)$. Thus after the M-step, Q is no longer equal to P.



(a) E-step (b) M-step

Figure 2.2.: Steps in the EM algorithm, adapted from [BN06]. In the first figure Q is set equal to P. In the second figure both the $\mathcal{L}$ and the $\mathbb{KL}$ increase.

2.2. Variational Inference (VI)

In the previous section $Q(Z)$ was set equal to $P(Z|Y)$. However, often this step is intractable because $P(Z|Y) = P(Z,Y)/P(Y)$ and we cannot evaluate the evidence

[2] $\mathcal{L}$ depends on θ as $Q(Z|\theta)$

term $P(Y) = \int P(Z,Y)dZ$. Instead, *Variational Inference* [JGJS99, WJ+08] uses the mean-field assumption. In this approximation, Q is chosen such that it factorizes into independent Q_is

$$Q(Z) = \prod_{i=1}^{M} Q_i(Z_i). \tag{2.10}$$

We want to find the distribution Q that maximizes the lower bound $\mathcal{L}(Q)$. Due to factorization, it is possible to do this in an iterative fashion. This can be shown by isolating $Q_j(Z_j)$ in the ELBO

$$\begin{aligned}
\mathcal{L}(Q) &= \mathop{\mathbb{E}}_{Q(Z)}\left[\log \frac{P(Z,Y)}{Q(Z)}\right] \\
&= \mathop{\mathbb{E}}_{Q(Z)}\left[\log P(Z,Y) - \log Q(Z)\right] \\
&= \mathop{\mathbb{E}}_{Q_j(Z_j)}\left[\mathop{\mathbb{E}}_{\sim Q_j(Z_j)}\left[\log P(Z,Y)\right] - \log Q_j(Z_j)\right] + \text{const} \\
&= -\mathbb{KL}(Q_j \| \mathop{\mathbb{E}}_{\sim Q_j(Z_j)}\left[\log P(Z,Y)\right]) + \text{const}
\end{aligned} \tag{2.11}$$

where $\mathbb{E}_{\sim Q_j(Z_j)}$ is the expectation w.r.t. to all factors except $Q_j(Z_j)$.

Suppose that we maximize $\mathcal{L}(Q)$ w.r.t. Q_j and keep all other Q_i fixed. The last line of (2.11) shows that this is equivalent to maximizing a negative Kullback-Leibler divergence between Q_j and $\mathbb{E}_{\sim Q_j(Z_j)}[\log P(Z,Y)]$, as the constant factor does not depend on Q_j. And so the optimal Q_j^* is

$$\log Q_j^*(Z_j) = \mathop{\mathbb{E}}_{\sim Q_j(Z_j)}\left[\log P(Z,Y)\right] + \text{const.} \tag{2.12}$$

The additive constant is the normalization constant, required to make sure Q^* is a valid distribution. The above equation states that the optimal solution for the factor $\log Q_j$ is obtained by setting it equal to the expected value, concerning all $\{Q_{i\neq j}\}$, of the log joint distribution of the hidden and visible variables.

2.3. Variational Message Passing (VMP)

Variational Message Passing [WBJ05] makes it easier and faster to apply factorized variational inference. It does so by using local computations and message passing on the graphical model.

2.3.1. Bayesian Networks

A *Probabilistic Graphical Model* (PGM) is a Bayesian Network that represents a set of variables and their conditional dependencies via a directed acyclic graph. Each node is a variable, and the directed edges between them indicate dependencies.

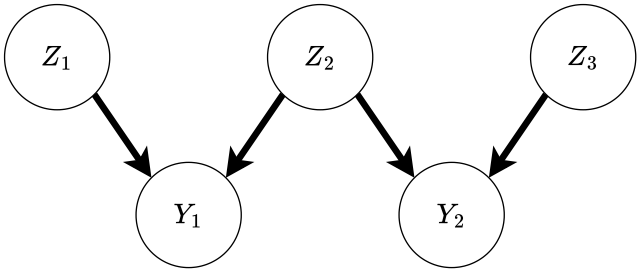


Figure 2.3.: A Bayesian Network

Take for example the following joint distribution

$$P(Y_1, Y_2, Z_1, Z_2, Z_3),$$

as in Figure 2.3. Then by the product rule this decomposes into

$$P(Y_1|Y_2, Z_1, Z_2, Z_3)P(Y_2|Z_1, Z_2, Z_3)P(Z_1, Z_2, Z_3).$$

Because of the properties of the Bayesian Network [3] this is equivalent to the following

$$P(Y_1|Z_1, Z_2)P(Y_2|Z_2, Z_3)P(Z).$$

Where the variables that are conditionally independent are eliminated. E.g. Y_1 is conditionally independent of Y_2 given Z_2, so we can omit Y_2 in the conditional $P(Y_1|Z_1, Z_2)$. In a more general manner, the joint distribution $P(Y)$ can be written as

$$P(Y) = \prod_i P(Y_i|\mathrm{pa}_i) \tag{2.13}$$

where pa_i stands for the parent of node i. For example, the parent nodes of Y_1 are Z_1 and Z_2.

2.3.2. Local updates

It is possible to do the variational inference updates using information from only local nodes. This also forms the connection between variational inference and bayesian networks.

Starting with equation (2.12) first generalize $P(Z, Y)$ to $P(X)$ i.e. X can now be both latent or observed variables. Then substitute the form of the joint probability distribution with $P(X) = \prod_i P(X_i|\mathrm{pa}_i)$:

$$\log Q_j^*(Z_j) = \mathop{\mathbb{E}}_{\sim Q_j(Z_j)}\left[\log \prod_k P(X_k|pa_k)\right] + \text{const} \tag{2.14}$$

[3] D-seperation [Pea88, BN06]

Any terms that do not depend on Q_j are included in the constant term. This term can be split into two, one which depends on the parents of Z_j, and one which depends on the children of Z_j

$$\begin{aligned}\log Q_j^*(Z_j) = &\mathop{\mathbb{E}}_{\sim Q_j(Z_j)}\Big[\log P(Z_j|pa_j)\Big] + \\ &\mathop{\mathbb{E}}_{\sim Q_j(Z_j)}\left[\log \prod_{k\in ch_j} P(X_k|pa_k)\right] + \text{const}\end{aligned} \tag{2.15}$$

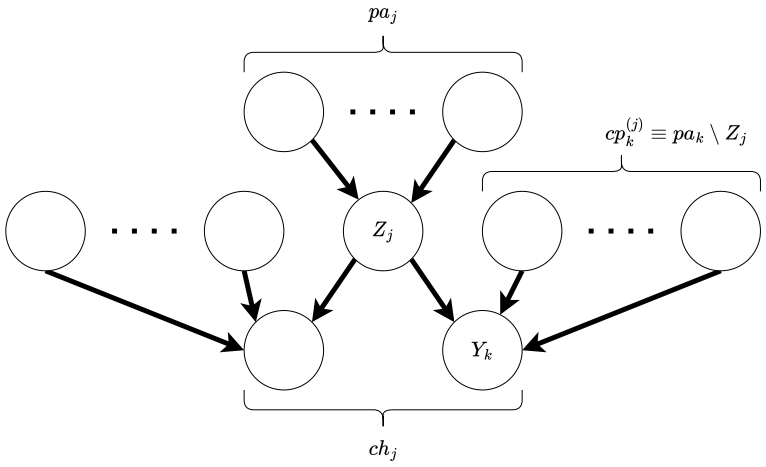


Figure 2.4.: Visialization of the Markov blanket, an adaptation of figure from [WBJ05]. The update w.r.t. Z_j depends on the parents pa_j and the children ch_j of Z_j. These terms are the messages from the corresponding nodes and reflect the local nature of the updates.

2.3.3. Conjugate-Exponential Models

In the case that the distributions are in the exponential family [4] there are important simplifications. For conjugate-exponential models, we have the following

$$\log P(Z|\mathrm{pa}_Z) = \eta_Z(\mathrm{pa}_Z)^T t_Z(Z) - A_Z(\eta_Z(\mathrm{pa}_Z)) \tag{2.16}$$

$$\log P(Y|Z,\mathrm{cp}_Z) = \eta_Y(Z,\mathrm{cp}_Z)^T t_Y(Y) - A_Y(\eta_Y(Z,\mathrm{cp}_Z)) \tag{2.17}$$

Because $P(Z|\mathrm{pa}_Z)$ is conjugate to $P(Y|Z,\mathrm{cp}_Z)$, it is possible to express $P(Y|Z,\mathrm{cp}_Z)$ in the same functional form w.r.t. Z as $P(Z|\mathrm{pa}_Z)$

$$\log P(Y|Z,\mathrm{cp}_Z) = \eta_{YZ}(Y,\mathrm{cp}_Z)^T t_Z(Z) - A_Z(\eta_{YZ}(Y,\mathrm{cp}_Z)) \tag{2.18}$$

[4] see Appendix C

Substituting the above equations into (2.15) it follows that

$$\begin{aligned}\log Q^*_Z(Z) = &\mathop{\mathbb{E}}_{\sim Q(Z)}\Big[\eta_Z(\mathrm{pa}_Z)^T t_Z(Z) - A_Z(\eta_Z(\mathrm{pa}_Z))\Big] \\ &+ \sum_{k\in \mathrm{ch}_Z} \mathop{\mathbb{E}}_{\sim Q(Z)}\Big[\eta_{YZ}(Y_k, \mathrm{cp}_k)^T t_Z(Z) - A_Z(\eta_{YZ}(Y_k, \mathrm{cp}_k))\Big] \\ &+ \mathrm{const}\end{aligned} \tag{2.19}$$

which is equivalent to

$$\begin{aligned}\log Q^*_Z(Z) = &\mathop{\mathbb{E}}_{\sim Q(Z)}\left[\eta_Z(\mathrm{pa}_Z) + \sum_{k\in \mathrm{ch}_Z} \eta_{YZ}(Y_k, \mathrm{cp}_k)\right]^T t_Z(Z) \\ &+ \mathrm{const}\end{aligned} \tag{2.20}$$

So we get that Q^*_Z is an exponential family distribution of the same form as $P(Z|\mathrm{pa}_Z)$ with natural parameters

$$\eta^*_Z = \mathop{\mathbb{E}}_{\sim Q(Z)}\left[\eta_Z(\mathrm{pa}_Z) + \sum_{k\in \mathrm{ch}_Z} \eta_{YZ}(Y_k, \mathrm{cp}_k)\right] \tag{2.21}$$

We can see in (2.17) and (2.18) that $P(Y|Z, \mathrm{cp}_Z)$ is linear w.r.t. to both $t_Y(Y)$ and $t_Z(Z)$ respectively, a property we will use next. Because of this linearity, it is possible to reparameterise these functions in terms of the expectations

$$\mathbb{E}\Big[\eta_Z(\mathrm{pa}_Z)\Big] = \tilde{\eta}_Z\Big(\{\mathbb{E}[t_i]\}_{i\in \mathrm{pa}_Z}\Big) \tag{2.22}$$

$$\mathbb{E}\Big[\eta_{YZ}(Y_k, \mathrm{cp}_k)\Big] = \tilde{\eta}_{YZ}\Big(\mathbb{E}[t_k], \{\mathbb{E}\Big[t_j\Big]\}_{j\in \mathrm{cp}_k}\Big) \tag{2.23}$$

The message from a parent node Z to a child node Y is the expectation under Q of the natural statistic vector

$$\mathbf{m}_{Z\to Y} = \mathbb{E}[t_Z] \tag{2.24}$$

The message from a child node Y to a parent node Z is

$$\mathbf{m}_{Y\to Z} = \tilde{\eta}_{YZ}\Big(\mathbb{E}[t_Y], \{\mathbf{m}_{i\to Y}\}_{i\in \mathrm{cp}_Z}\Big) \tag{2.25}$$

If node X is observed the expectation $\mathbb{E}[t_X]$ is replaced by t_X. So we get that the natural parameters in (2.21) can be computed using local updates shown in (2.24), (2.25)

$$\eta^*_Z = \tilde{\eta}_Z\Big(\{\mathbf{m}_{i\to Z}\}_{i\in \mathrm{pa}_Z}\Big) + \sum_{j\in \mathrm{ch}_Z} \mathbf{m}_{j\to Z} \tag{2.26}$$

2.4. Variational Autoencoder (VAE)

The *Variational Autoencoder* [KW13] was developed with the aim of performing efficient approximate inference and learning with directed probabilistic models whose continuous latent variables and/or parameters have intractable posterior distributions.

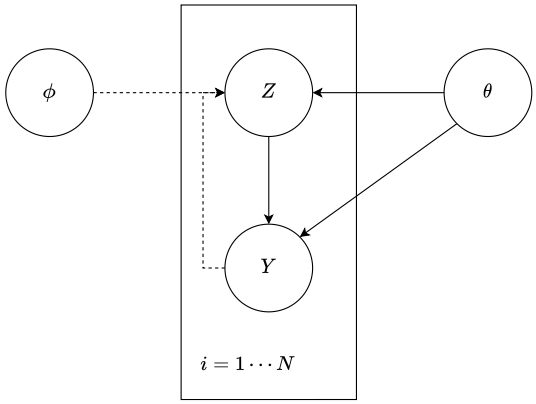


Figure 2.5.: Graphical representation of generative model denoted by the solid lines. The dashed lines indicate the variational approximation.

Recall in (2.4) where $Q(Z|\theta)$ was introduced. Instead, here we will instead introduce $Q(Z|Y,\phi)$. In the following the parameters θ and ϕ are denoted by a subscript

$$\begin{aligned}
\log P_\theta(Y) &= \mathop{\mathbb{E}}_{Q_\phi(Z|Y)}\Big[\log P_\theta(Z,Y) - \log Q_\phi(Z|Y)\Big] \\
&\quad - \mathop{\mathbb{E}}_{Q_\phi(Z|Y)}\Big[\log P_\theta(Z|Y) - \log Q_\phi(Z|Y)\Big] \\
&= \mathop{\mathbb{E}}_{Q_\phi(Z|Y)}\left[\log \frac{P_\theta(Z,Y)}{Q_\phi(Z|Y)}\right] - \mathop{\mathbb{E}}_{Q_\phi(Z|Y)}\left[\log \frac{P_\theta(Z|Y)}{Q_\phi(Z|Y)}\right] \\
&= \mathcal{L}(Q;\theta,\phi) + \mathbb{KL}(Q_\phi||P_\theta(Z|Y)) \\
&\geq \mathcal{L}(Q;\theta,\phi).
\end{aligned} \tag{2.27}$$

We can decompose the ELBO into a reconstruction and regularizer term as follows

$$\begin{aligned}
\mathcal{L}(Q;\theta,\phi) &= \mathop{\mathbb{E}}_{Q_\phi(Z|Y)}\left[\log \frac{P_\theta(Z,Y)}{Q_\phi(Z|Y)}\right] \\
&= \mathop{\mathbb{E}}_{Q_\phi(Z|Y)}\left[\log \frac{P_\theta(Y|Z)P(Z)}{Q_\phi(Z|Y)}\right] \\
&= \mathop{\mathbb{E}}_{Q_\phi(Z|Y)}\left[\log P_\theta(Y|Z)\right] - \mathbb{KL}(Q_\phi||P_\theta(Z))
\end{aligned} \tag{2.28}$$

The Monte Carlo estimator for this objective is

$$\mathcal{L}(Q;\theta,\phi) \approx \frac{1}{L}\sum_{l=1}^{L} \log P_\theta(Y,Z^{(l)}) - \log Q_\phi(Z^{(l)}|Y), \tag{2.29}$$

where $Z \sim Q_\phi(Z|Y)$. However, this has high variance when used for gradients [PBJ12]. Instead we can use the reparameterization trick, and use $\tilde{Z} = g_\phi(\epsilon, x)$, with $\epsilon \sim p(\epsilon)$. Then we can use the following Monto Carlo estimator

$$\mathcal{L}(Q;\theta,\phi) \approx \frac{1}{L}\sum_{l=1}^{L} \log P_\theta(Y,g_\phi(\epsilon^{(l)},Y)) - \log Q_\phi(g_\phi(\epsilon^{(l)},Y)|Y), \tag{2.30}$$

For example, in the case Q is a Gaussian distribution we get

$$Z = \mu + \sigma \odot \epsilon \text{ and } \epsilon \sim \mathcal{N}(0,I) \tag{2.31}$$

2.4.1. Encoder-Decoder architecture

In practice, we can use Multilayer Perceptrons (MLPs) to model the encoder and decoder. The encoder models the Gaussian $Q_\phi(Z)$ and outputs a mean μ and variance Σ such that we get

$$Q_\phi(Z) = \mathcal{N}(\mu(Y;\phi), \Sigma(Y;\phi)) \tag{2.32}$$

Then the latents Z are obtained using the reparameterization trick (2.31). Finally, the decoder models the log-likelihood $P_\theta(Y|Z)$, such that we get

$$P_\theta(Y|Z) = \mathcal{N}(\mu(Z;\theta), \Sigma(Z;\theta)) \tag{2.33}$$

This model can be trained end-to-end, where the gradients for the encoder and decoder can be found by automatic differentiation on $\mathcal{L}(Q;\theta,\phi)$.

3. Structured Variational Autoencoder (SVAE)

The *Structured Variational Autoencoder* [JDW+16] is a composition of a probabilistic graphical model and a neural network. This composition aims to achieve both the flexibility of a neural network and the structure of a probabilistic graphical model. One way of looking at this model is as a generalization of the variational autoencoder (section 2.2).

As in the variational autoencoder, the SVAE uses the observations Y to obtain approximations for the latent variables. Additionally, the local latent variables X have additional structure, such as clusters of Gaussians (section 4.1). These local latent variables X also require optimization, which is done using variational message passing (section 2.3). To do efficient local optimization the distributions should be in the exponential family. However, this is not the case if a neural network is used to approximate the latent variables. To fix this issue a trick is used, which is represented by ϕ.

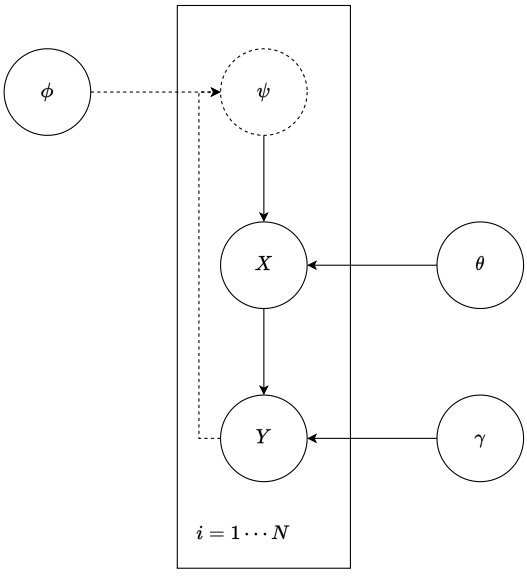


Figure 3.1.: Graphical representation of the structured variational autoencoder.

In the next sections, we will first look at the SVAE objective, and how we get there using partial optimization of the local latent variables (section 3.1). Second, using

the SVAE objective we can obtain the natural gradients used to optimize the global variables (section 3.2). Thirdly, by automatic differentiation w.r.t. the SVAE objective the parameters of the encoder and decoder are updated (section 3.3).

3.1. Optimizing the local latents

Let $p(x|\theta)$ be an exponential family and let $p(\theta)$ be its corresponding conjugate prior, i.e.

$$p(\theta) = \exp\{\langle \eta_\theta, t_\theta(\theta)\rangle - \log Z_\theta(\eta_\theta^0)\} \tag{3.1}$$

$$p(x|\theta) = \exp\{\langle \eta_x(\theta), t_x(x)\rangle - \log Z_x(\eta_x^0)\} \tag{3.2}$$

Let $p(y|x,\gamma)$ be a general family of densities, with the exponential family prior on its parameters $p(\gamma)$. For fixed y consider the mean field family of densities $q(\theta,\gamma,x) = q(\theta)q(\gamma)q(x)$ and the mean-field variational inference objective

$$\mathcal{L}(q(\theta)q(\gamma)q(x)) \triangleq \mathop{\mathbb{E}}_{q(\theta)q(\gamma)q(x)}\Big[\log \frac{p(\theta)p(\gamma)p(x|\theta)p(y|x,\gamma)}{q(\theta)q(\gamma)q(x)}\Big] \tag{3.3}$$

We take $q(\theta)$, $q(\gamma)$, $q(x)$ to be in the same exponential family as $p(\theta)$, $p(\gamma)$, $p(x|\theta)$ respectively; with natural parameters η_θ, η_γ, and, η_x. We end up with the following mean field variational inference objective

$$\mathcal{L}(\eta_\theta,\eta_\gamma,\eta_x) \triangleq \mathop{\mathbb{E}}_{q(\theta)q(\gamma)q(x)}\Big[\log \frac{p(\theta)p(\gamma)p(x|\theta)p(y|x,\gamma)}{q(\theta)q(\gamma)q(x)}\Big] \tag{3.4}$$

To perform efficient optimization we choose the variational parameter η_x as a function of η_θ and η_γ. We could set η_x to be a local partial optimizer of $\mathcal{L}$. However, optimizing over general densities $p(y|x,\gamma)$ can be computationally expensive. Instead we choose η_x to be a optimizer over a surrogate objective $\hat{\mathcal{L}}$

$$\begin{aligned}\hat{\mathcal{L}}(\eta_\theta,\eta_\gamma,\eta_x,\phi) &\triangleq \mathop{\mathbb{E}}_{q(\theta)q(\gamma)q(x)}\Big[\log \frac{p(\theta)p(\gamma)p(x|\theta)\exp\psi(x;y,\phi)}{q(\theta)q(\gamma)q(x)}\Big] \\ &= \mathop{\mathbb{E}}_{q(\theta)q(x)}\Big[\log \frac{p(\theta)p(x|\theta)\exp\psi(x;y,\phi)}{q(\theta)q(x)}\Big] + \text{const}\end{aligned} \tag{3.5}$$

where we pulled terms involving γ into the constant which does not depend on η_x. The function $\psi(x;y,\phi)$ is defined in a form related to the exponential family:

$$\psi(x;y,\phi) \triangleq r(y;\phi)^T t_x(x), \tag{3.6}$$

and fits into the conjugate structure. The function r is called the *recognition model*, e.g. an MLP. We can find η_x as a partial local optimizer

$$\eta_x^*(\eta_\theta,\phi) \triangleq \arg\max_{\eta_x} \hat{\mathcal{L}}(\eta_\theta,\eta_\gamma,\eta_x,\phi) \tag{3.7}$$

Given this η_x^* the SVAE objective is defined as

$$\mathcal{L}_{\text{SVAE}}(\eta_\theta, \eta_\gamma, \phi) \triangleq \mathcal{L}(\eta_\theta, \eta_\gamma, \eta_x^*(\eta_\theta, \phi)) \tag{3.8}$$

and the related optimization problem we want to solve

$$\max_{\eta_\theta, \eta_\gamma, \phi} \mathcal{L}_{\text{SVAE}}(\eta_\theta, \eta_\gamma, \phi) \tag{3.9}$$

Without additional conjugacy structure, see Section A.2, these local optimizations can be performed with generic gradient-based methods and automatic differentiation [DA15].

3.2. Natural gradients of global parameters

Assume we have the following objective, as in (stochastic) variational inference [HBWP13]:

$$\mathcal{L}(\eta_\theta) \triangleq \mathop{\mathbb{E}}_{q(\theta)q^*(x)}\left[\log \frac{p(\theta)p(x,y|\theta)}{q(\theta)q^*(x)}\right] \tag{3.10}$$

for which we want to obtain the gradients w.r.t. to the global parameters θ. For this, the conjugacy property of the exponential family is used, which is shown in Proposition 1.

Proposition 1. *Let the densities $p(\theta)$ and $p(x|\theta)$ be defined as in* (3.1) *and* (3.2)*. We have the relations*

$$p(\theta, x) = \exp\{\langle \eta_\theta + (t_x(x), 1), t_\theta(\theta)\rangle - \log Z_\theta(\eta_\theta)\} \tag{3.11}$$

$$p(\theta|x) = \exp\{\langle \eta_\theta + (t_x(x), 1), t_\theta(\theta)\rangle - \log Z_\theta(\eta_\theta + (t_x(x), 1)\} \tag{3.12}$$

and hence in particular the posterior $p(\theta|x)$ is in the same exponential family as $p(\theta)$ with the natural parameter $\eta_\theta + (t_x(x), 1)$. Similarly with multiple likelihood terms $p(x_i|\theta)$ for $i = 1, 2, \cdots, N$ we have

$$p(\theta)\prod_{i=1}^{N} p(x_i|\theta) = \exp\{\langle \eta_\theta + \sum_{i=1}^{N}(t_x(x_i), 1), t_\theta(\theta)\rangle - \log Z_\theta(\eta_\theta)\} \tag{3.13}$$

From the conjugacy property, it follows that the posterior $p(\theta|x)$ is in the same exponential family as the prior $p(\theta)$. The only difference between the prior and posterior are the natural parameters. This property is important for the proof of Proposition 2, which states the gradient.

Proposition 2. *Let the objective $\mathcal{L}(\eta_\theta)$ be defined as in* (3.10)*. Then the gradient $\nabla\mathcal{L}(\eta_\theta)$ is*

$$\nabla^2 \log Z_\theta(\eta_\theta)\left(\eta_\theta^0 + \mathop{\mathbb{E}}_{q^*(x)}\left[(t_{xy}(x, y), 1)\right] - \eta_\theta\right) \tag{3.14}$$

where $q^*(x)$ is a local partial optimizer of the mean-field objective for fixed global variational parameters η_θ.

Proof. Because $q^*(x)$ is a local partial optimizer of the mean-field objective we have that

$$\nabla \mathcal{L}(\eta_\theta) = \nabla_{\eta_\theta} \mathop{\mathbb{E}}_{q(\theta)q^*(x)} \left[\log \frac{p(\theta)p(x,y|\theta)}{q(\theta)q^*(x)} \right] \tag{3.15}$$

Using Proposition 1 we can write the right-hand side without the gradient as

$$\mathop{\mathbb{E}}_{q(\theta)q^*(x)} \left[\log \frac{\exp\{\langle \eta_\theta^0 + (t_{xy}(x,y),1), t_\theta(\theta)\rangle - \log Z_\theta(\eta_\theta^0)\}}{\exp\{\langle \eta_\theta, t_\theta(\theta)\rangle - \log Z_\theta(\eta_\theta)\}} \right] \tag{3.16}$$

by plugging in the exponential family definitions. This expression is equivalent to

$$\langle \eta_\theta^0 + \mathop{\mathbb{E}}_{q^*(x)} \Big[(t_{xy}(x,y),1) \Big] - \eta_\theta, \mathop{\mathbb{E}}_{q_\theta} [t_\theta(\theta)] \rangle - (\log Z_\theta(\eta_\theta^0) - \log Z_\theta(\eta_\theta)) \tag{3.17}$$

Using $\mathbb{E}_{q(\theta)}[t_\theta(\theta)] = \nabla \log Z_\theta(\eta_\theta)$ (Appendix C.4) we get

$$\nabla \log Z_\theta(\eta_\theta) \left(\eta_\theta^0 + \mathop{\mathbb{E}}_{q^*(x)} \Big[(t_{xy}(x,y),1) \Big] - \eta_\theta \right) - (\log Z_\theta(\eta_\theta^0) - \log Z_\theta(\eta_\theta)) \tag{3.18}$$

and we get the final result by then taking the gradient w.r.t. η_θ and using the product rule:

$$\nabla^2 \log Z_\theta(\eta_\theta) \left(\eta_\theta^0 + \mathop{\mathbb{E}}_{q^*(x)} \Big[(t_{xy}(x,y),1) \Big] - \eta_\theta \right) - \log Z_\theta(\eta_\theta) + \log Z_\theta(\eta_\theta) \tag{3.19}$$

□

3.2.1. Natural Gradients of the SVAE

In the previous section, we looked at a VI objective without non-linear observations. The gradients w.r.t. the global parameters for this objective can be extended for the SVAE objective (3.8).

$$\begin{aligned} \nabla_{\eta_\theta} \mathcal{L}_{\text{SVAE}} = & \nabla^2 \log Z_\theta(\eta_\theta) \left(\eta_\theta^0 + \mathop{\mathbb{E}}_{q^*(x)} \Big[(t_{xy}(x,y),1) \Big] - \eta_\theta \right) \\ & + \nabla_{\eta_\theta} \eta_x^*(\eta_\theta, \phi) \left(\nabla_x \mathcal{L}(\eta_\theta, \eta_\gamma, \eta_x^*(\eta_\theta, \phi)) \right) \end{aligned} \tag{3.20}$$

This follows from the chain rule[1] applied to the SVAE objective. The first term was derived in the previous section. In the case of variational inference, the second part is zero. This is not the case for the SVAE, but it can be estimated using the reparameterization trick.

The natural gradient [Ama98] is defined by

$$\tilde{\nabla} \mathcal{L}(\eta_\theta) \triangleq (\nabla^2 \log Z_\theta(\eta_\theta))^{-1} \nabla \mathcal{L}(\eta_\theta) \tag{3.21}$$

[1] chain rule on $g(x) \triangleq f(x, y^*(x))$ gives $\nabla g(x) = \nabla_x f(x, y^*(x)) + \nabla y^*(x) \nabla_y f(x, y^*(x))$

where $\nabla^2 \log Z_\theta(\eta_\theta)$ is the *Fisher information matrix*. In the context of the SVAE, natural gradient descent is effectively a second-order optimization problem. In general, second-order optimization problems provide faster convergence compared to first-order optimization problems. Often this faster convergence comes at a higher cost of computing the second-order gradients. For the second term, we do need to compute the Fisher information matrix. However, when η_θ is small in dimension compared to the other parameters this is not much extra computation.

3.3. Gradients of the autoencoder parameters

To compute an unbiased stochastic estimate of the gradients we can use the reparameterization trick as in Section 2.4. We rewrite the $\mathcal{L}_{\text{SVAE}}$ objective into a term that requires this sample-based approximation and one that can be computed directly.

$$\begin{aligned}\mathcal{L}_{\text{SVAE}}(\eta_\theta, \eta_\gamma, \phi) = &\mathop{\mathbb{E}}_{q(\gamma)q^*(x)}\Big[\log p(y|x,\gamma)\Big] \\ &- \mathbb{KL}(q(\theta)q(\gamma)q^*(x)\|p(\theta,\gamma,x)).\end{aligned} \tag{3.22}$$

Here the first term needs to be estimated using the reparameterization trick.

Let $\hat{\gamma}(\eta_\gamma) \sim q(\gamma)$ and $\hat{x}(\phi) \sim q^*(x)$. Then unbiased estimates of the gradients are given by

$$\begin{aligned}\nabla_\phi \mathcal{L}_{\text{SVAE}}(\eta_\theta, \eta_\gamma, \phi) \approx &\nabla_\phi \log p(y|\hat{x}(\phi), \hat{\gamma}(\eta_\gamma)) \\ &- \nabla_\phi \mathbb{KL}(q(\theta)q^*(x)\|p(\theta,x))\end{aligned} \tag{3.23}$$

$$\begin{aligned}\nabla_{\eta_\gamma} \mathcal{L}_{\text{SVAE}}(\eta_\theta, \eta_\gamma, \phi) \approx &\nabla_{\eta_\gamma} \log p(y|\hat{x}(\phi), \hat{\gamma}(\eta_\gamma)) \\ &- \nabla_{\eta_\gamma} \mathbb{KL}(q(\gamma)\|p(\gamma))\end{aligned} \tag{3.24}$$

which we can compute by doing automatic differentiation over the Monto-Carlo estimate of

$$\begin{aligned}\mathcal{L}_{\text{SVAE}}(\eta_\theta, \eta_\gamma, \phi) \approx &\log p(y|\hat{x}(\phi), \hat{\gamma}(\eta_\gamma)) \\ &- \mathbb{KL}(q(\theta)q(\gamma)q^*(x)\|p(\theta,\gamma,x))\end{aligned} \tag{3.25}$$

4. Graphical Models

Graphical models[1] are an alternative representation of a joint distribution. A graphical model represents a set of variables and their conditional dependencies via a directed acyclic graph (section 2.3.1). Each node is a variable, and the directed edges between them indicate dependencies. This representation can help with operations such as Variational Message Passing (section 2.3). In addition, we can use it to construct models, which is done in the next sections.

4.1. Gaussian Mixture Model (GMM)

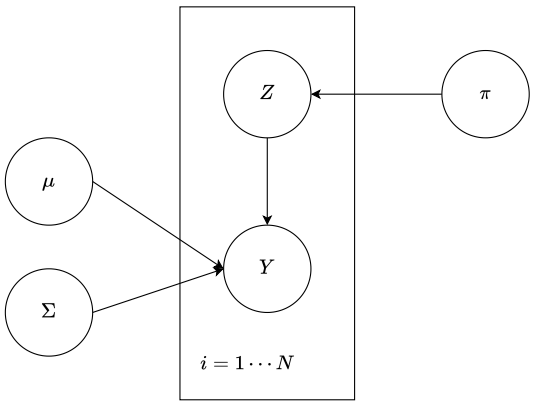


Figure 4.1.: Graphical representation of the Gaussian Mixture Model, following the generative model in equation (4.1). Each datapoint Y_i belongs to a cluster k, which is determined by the (local) latent variable Z_i. The (global) cluster parameters μ_k and Σ_k are shared across all data points.

A *Gaussian Mixture Model* (GMM) is used to model several distinct "clusters" of data points. It is possible to observe the data points, but not which cluster they belong to. Each cluster is modeled using a Gaussian distribution, with a latent[2] variable

[1] In this context a graphical model refers to a probabilistic graphical model.
[2] A latent variable is not observed.

determining which cluster a data point belongs to.

$$\begin{aligned} \pi &\sim \text{Dir}(\alpha) \\ (\mu_k, \Sigma_k) &\overset{iid}{\sim} \text{NIW}(\lambda) \\ z_n|\pi &\overset{iid}{\sim} \pi \\ y_n|z_n, \gamma, \{(\mu_k, \Sigma_k)\}_{k=1}^K &\overset{iid}{\sim} \mathcal{N}(\mu_{z_n}, \Sigma_{z_n}) \end{aligned} \tag{4.1}$$

For the GMM the graphical model is shown in Figure 4.1, using the equations in (4.1). These equations are also called the generative model and describe how a data point is generated. Every data point y_n is sampled from a Gaussian distribution with parameters (μ_k, Σ_k) depending on which cluster k it belongs to. The cluster assignments k are sampled from a Categorical distribution with parameters Z_i. The priors for the Gaussian and Categorical are the Normal Inverse Wishart and Dirichlet distributions respectively. The parameters of the GMM can be optimized using the EM algorithm (see section 2.1).

$$\begin{aligned} \gamma &\sim p(\gamma) \\ x_n &\overset{iid}{\sim} \mathcal{N}(0, I) \\ y_n|x_n, \gamma &\overset{idd}{\sim} \mathcal{N}(\mu(x_n; \gamma), \Sigma(x_n; \gamma)) \end{aligned} \tag{4.2}$$

The model above is inflexible and assumes that the observations y_n are Gaussian. In equation (4.2), a generative model is shown that assumes a non-linear function, such as a neural network. Here $\mu(x_n; \gamma)$ and $\Sigma(x_n; \gamma)$ depend on x_n, trough some non-linear function. The parameters are optimized using the maximum likelihood principle (see the VAE in section 2.4).

$$\begin{aligned} \pi &\sim \text{Dir}(\alpha) \\ (\mu_k, \Sigma_k) &\overset{idd}{\sim} \text{NIW}(\lambda) \\ \gamma &\sim p(\gamma) \\ z_n|\pi &\overset{idd}{\sim} \pi \\ x_n &\overset{idd}{\sim} \mathcal{N}(\mu^{(z_n)}, \Sigma^{(z_n)}) \\ y_n|x_n, \gamma &\overset{idd}{\sim} \mathcal{N}(\mu(x_n; \gamma), \Sigma(x_n; \gamma)) \end{aligned} \tag{4.3}$$

The generative model in (4.2) is model flexible in modeling the observations y_n, but it does not model the latent variables z_n. By creating a composition of the GMM with non-linear observations it is possible to combine the strengths of both. The combination of the GMM with a neural network is the SVAE-GMM, the generative model is shown in (4.3). The SVAE-GMM is flexible in modeling the density of the observations, and it models the latent variables.

4.2. Linear Dynamical System (LDS)

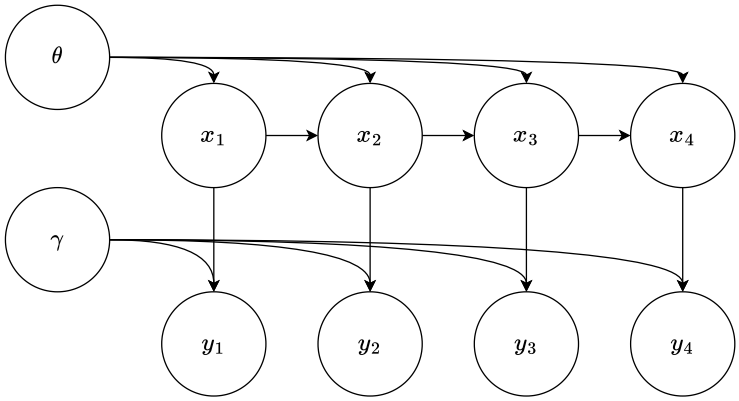


Figure 4.2.: Graphical representation for the Linear Dynamical System. The observations y_i are dependent on the (local) latent states x_i. The latent states x_i are dependent on the previous latent states x_{i-1}. The (global) parameters γ and θ parametrize these relations. In this figure only four time steps are shown, this can be extended to any number of time steps.

A *Linear Dynamical System* (LDS) is a model that is used to model time series data, where observations are dependent on the previous observations. The graphical model is shown in Figure 4.2.

$$\begin{aligned} \mu_1, Q_1 &\sim \text{NIW}(\nu, \Phi, \kappa, \mu_0) \\ A, Q &\sim \text{MNIW}(\nu, \Phi, K, M) \\ x_1 &\sim \mathcal{N}(\mu_1, Q_1) \\ x_t &\sim \mathcal{N}(Ax_{t-1}, Q) \\ y_t | x_t &\sim \mathcal{N}(Cx_t, R) \end{aligned} \tag{4.4}$$

The generative model is shown in (4.4). An observation y_t is sampled from a Gaussian distribution with parameters C and R, dependent on the latent state x_t. The latent state x_t is sampled from a Gaussian distribution with parameters A and Q, dependent on the previous latent state x_{t-1}. The transition and initial parameters use the Matrix Normal Inverse Wishart and the Normal Inverse Wishart distribution as a prior respectively.

$$
\begin{aligned}
\mu_1, Q_1 &\sim \text{NIW}(\nu, \Phi, \kappa, \mu_0) \\
A, Q &\sim \text{MNIW}(\nu, \Phi, K, M) \\
\gamma &\sim p(\gamma) \\
x_1 &\sim \mathcal{N}(\mu_1, Q_1) \\
x_t &\sim \mathcal{N}(Ax_{t-1}, Q) \\
y_t | x_t, \gamma &\sim \mathcal{N}(\mu(x_t; \gamma), \Sigma(x_t; \gamma))
\end{aligned}
\tag{4.5}
$$

As in the previous section for the GMM, it is possible to replace the linear observation in $\mathcal{N}(Cx_t, R)$ with a non-linear observation, $\mathcal{N}(C(x_t; \gamma), R(x_t; \gamma))$ where the parameters of the Gaussian depend on a parameterized function, such as a neural network. This results in the SVAE-LDS.

4.3. More models

We have seen two different instantiations of the SVAE, the SVAE-GMM and the SVAE-LDS. These are just two examples, and there exist many more possible compositions. In the next section it will be shown how it is conceptually simple to create a new composition, we "just"[3] need to derive new local updates.

[3] In practice this can be non-trivial. In addition, other changes need to be done, such as data preparation.

5. Methods

In the following section, we will dissect the (pseudo) code [1] of the SVAE. Note that the code displayed here is somewhat simplified.

The SVAE algorithm is a culmination of the previous sections and is displayed in Algorithm 1. In step 1, the encoder r is used to predict the potentials ψ. These potentials are used to perform inference (see section 2.3). The results of the inference step are the latent variables $\hat{x}$, statistics $\bar{t}_x$, and local $\mathbb{KL}$. These quantities are used to compute the gradients for the encoder and decoder (see section 2.4.1), and the natural gradient for the global variables.

Algorithm 1: Estimate SVAE lower bound and its gradients

Input: parameters $(\eta_\theta, \eta_\gamma, \phi)$ and data sample y

function: SVAEGradients $(\eta_\theta, \eta_\gamma, \phi, y)$

$\psi \leftarrow r(y_n; \phi)$

$(\hat{x}, \bar{t}_x, \mathbb{KL}^{\text{local}}) \leftarrow \text{PGMInference}(\eta_\theta, \psi)$

$\hat{\gamma} \sim q(\gamma)$

$\mathcal{L} \leftarrow N \log p(y|\hat{x}, \hat{\gamma}) - N\,\mathbb{KL}^{\text{local}} - \mathbb{KL}(q(\theta)q(\gamma)\|p(\theta)p(\gamma))$

$\tilde{\nabla}_{\eta_\theta}\mathcal{L} \leftarrow \eta_\theta^0 - \eta_\theta + N(\bar{t}_x, 1) + N(\nabla_{\eta_x} \log p(y|\hat{x}, \hat{\gamma}), 0)$

return: lower bound $\mathcal{L}$, natural gradient $\tilde{\nabla}_{\eta_\theta}\mathcal{L}$, gradients $\nabla_{\eta_\gamma, \phi}\mathcal{L}$

function: PGMInference(η_θ, ψ)

$q^*(x) \leftarrow \text{OptimizeLocalFactors}(\eta_\theta, \psi)$

return: sample $\hat{x} \sim q^*(x)$, statistics $\mathbb{E}_{q^*(x)}\, t_x(x)$, divergence $\mathbb{E}_{q(\theta)}\, \mathbb{KL}(q^*(x)\|p(x|\theta))$

The code is implemented using Python and PyTorch [PGM+19]. In the next sections, we will look at a small part of this implementation. First at the `PGMInference` function for the SVAE-GMM and the SVAE-LDS, then return to the main SVAE training loop.

5.1. SVAE-GMM

Recall that the GMM is a mixture model, and both the cluster assignment and the cluster parameters need to be optimized. This cannot be done in a single optimization

[1] Full Python code is available at `github.com/JohanvandenHeuvel/SVAE`

step, as the cluster assignment depends on the cluster parameters and vice versa. Which implies that an inner optimization loop is required.

The function `local_optimization` shows this inner optimization loop. The loop contains a step to optimize the cluster parameters (μ, Σ) and a step to optimize the cluster assignments z (for more detail about the contents of these functions see appendix A.2). As shown in Algorithm 1, this function takes in the potentials ψ and the global parameters η_θ. The function also returns latents $\hat{x}$, statistics $\bar{t}_x$ and the local $\mathbb{KL}$, which is not shown in the code snippet.

```
def local_optimization(potentials, global_param):
  label_stats = initialize_meanfield(label_parameters, potentials).double
                                                    ()
  for i in range(epochs):
    # Gaussian x
    _, gaussian_stats, gaussian_kld = gaussian_optimization(
        gaussian_parameters, potentials, label_stats
    )
    # Label z
    _, label_stats, label_kld = label_optimization(
        gaussian_parameters, label_parameters, gaussian_stats
    )
```

The global parameters of the GMM are the parameters of the Dirichlet distribution and the Normal Inverse Wishart distribution, as these are the prior distributions. These global parameters are updated using the natural gradient (see section 3.2).

5.2. SVAE-LDS

The local parameters of the SVAE-LDS are the latent nodes x_t. Messages from the parent node x_{t-1} and child nodes x_{t+1} and y_t are required to update node x_t using message passing. However, messages from the parents depend on their parents and messages from the children on their children.

Because of this dependence, there are two steps to updating the local parameters. First, all the (forward) messages from the parents are calculated. Second, do the inverse and calculate the (backward) messages from the children. During the second step, also update the local parameters. See more about Kalman filtering, smoothing, and sampling in the appendix; sections B.2, B.3, and B.4 respectively.

```
def local_optimization(potentials, global_param):
  forward_messages, logZ = info_kalman_filter(
      init_params=init_param, pair_params=(J11, J12, J22, logZ),
                                          observations=y
  )
  _, expected_stats = info_kalman_smoothing(
      forward_messages, pair_params=(J11, J12, J22)
  )
  samples = info_sample_backward(
      forward_messages, pair_params=(J11, J12, J22), n_samples=n_samples
  )
```

The global parameters of the SVAE-LDS are the parameters for the priors of the transition mechanics A, Q and the initial condition μ_1, Q_1. The prior distributions are the Matrix Normal Inverse Wishart and Normal Inverse Wishart respectively.

5.3. SVAE

Returning to the main SVAE training loop, the Python implementation of Algorithm 1 is shown in the code below.

This code is similar to an implementation of the VAE (see 2.4.1). However, here in addition the natural gradient is computed, as described in section 3.2. Note that this code is the same for different models, such as the GMM and LDS discussed above. The main difference lies in how `local_optimization` is implemented.

```
global_prior = init_global_params()
global_param = init_global_params()
optimizer = Adam()
global_optimizer = SGD()
for epoch in range(epochs):
  for y in dataset:
    potentials = encoder(y)
    x, stats, local_kld = local_optimizer(potentials, global_param)
    nat_grad = natural_gradient(stats, global_param, global_prior)
    eta_theta = global_optimizer.update(global_param, nat_grad)
    mu_y, log_var_y = decoder(x)
    global_kld = kld(global_param, global_prior)
    loss = reconstruction_loss(y, mu_y, log_var_y) + local_kld +
                                          global_kld
    optimizer.update(encoder, decoder, loss)
```

6. Results

In the following section, empirical experiments are done to investigate the behavior of the SVAE. For this purpose, two different graphical models are used. The first one is a Gaussian Mixture Model (section 4.1), used to model 2D point data. The second one is a Linear Dynamical System (section 4.2), used for a 1D time series. For

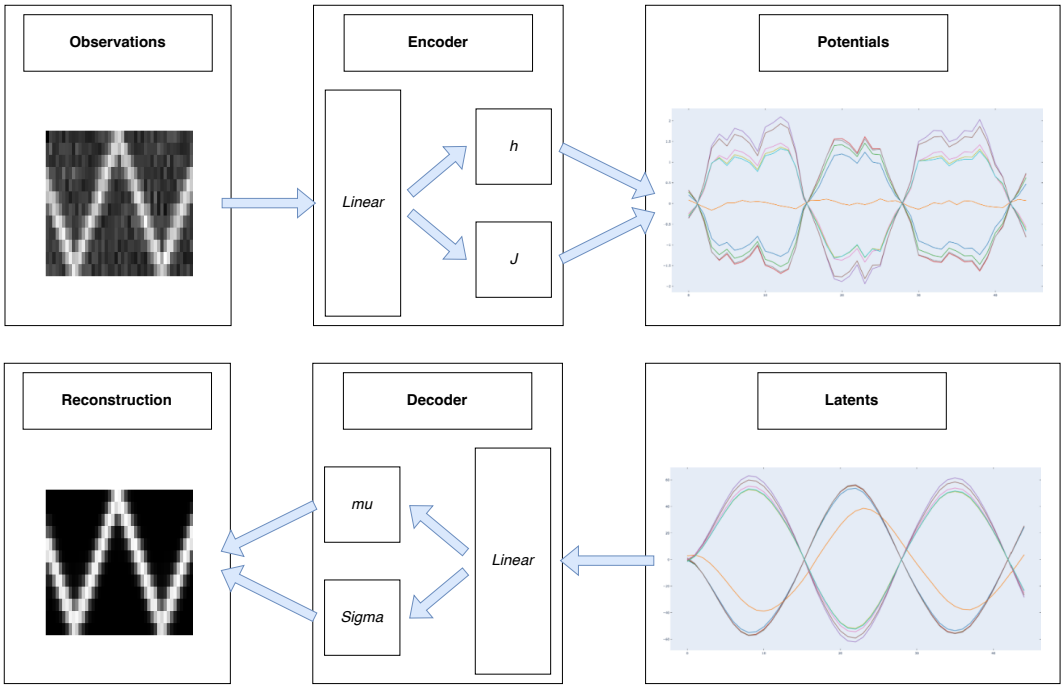
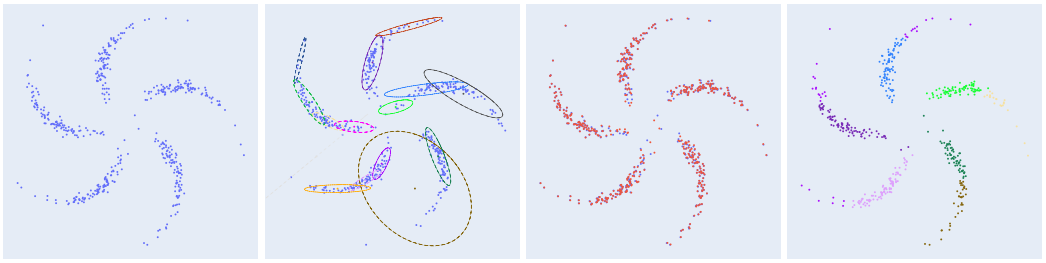


Figure 6.1.: SVAE-LDS showing the general SVAE setup.

the encoder and decoder (section 2.4.1), fully connected neural networks are used. The encoder consists of a single hidden layer of size 50, which splits into two layers of size 50, one for the mean and one for the variance of the potentials. The decoder also consists of a single hidden layer of size 50, which splits into two layers of size 50, one for the mean and one for the variance of the reconstruction. The encoder and decoder weights are initialized using samples from $\mathcal{N}(0, 1e{-}2)$. The activation functions used are ReLUs. Specific details of the models, such as the data generation, are described in their respective sections.



(a) Observations (b) GMM (c) VAE (d) SVAE

Figure 6.2.: Figure showing the different approaches to model 2D synthetic spiral data. **(a):** the observations we want to model. **(b):** the GMM requires multiple clusters to model the spirals. This is because the Gaussian shape is inflexible. **(c):** the VAE models the shape well. However, data points are not assigned to clusters. **(d):** the SVAE is also able to model the shape well. In addition, data points are assigned to clusters, which are indicated by the color of the points.

6.1. SVAE-GMM

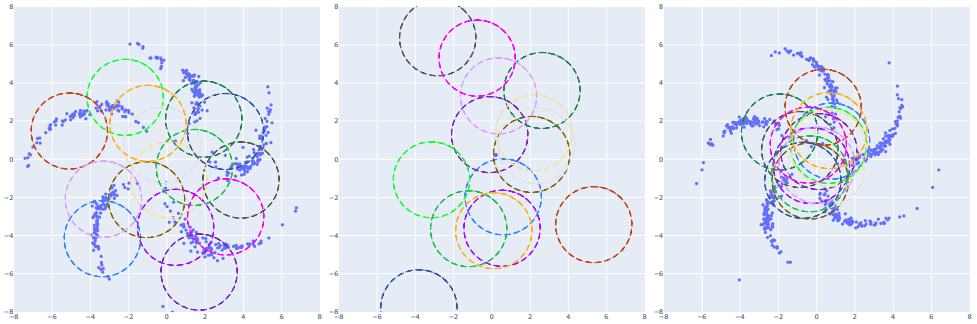
For the GMM residual connections are used together with the encoder and decoder. The residual connections are linear layers going directly from the input to the latent dimension, and the latent dimension to the output dimension. As the residual connections are doing linear regression they learn faster than the encoder or decoder. This is important for getting a good starting point. For the experiments, synthetic data is created. The data consists of five different spirals (Figure 6.2a). These different spirals are distinct clusters, and the expectation is that a model assigns data points in the same spiral to the same cluster.

The GMM requires multiple clusters to model the spirals (Figure 6.2b), as Gaussians are quite inflexible in shape. The VAE can model the shape of the spirals (Figure 6.2c) but does not assign data points to clusters. The SVAE-GMM combines their strengths and both models the shape of the spirals and assigns data points to clusters (Figure 6.2d).

This shows that the SVAE-GMM can combine both modeling strengths. However, it still has the weakness of the GMM, that it not always converges to the solution we would expect.

6.1.1. Initialization of the SVAE-GMM

The initial conditions for the SVAE-GMM are very important in how likely it is that the algorithm converges to a good solution. An intuitive way of understanding this is as follows. At each iteration, the SVAE-GMM optimizes the local variables. This local optimization consists of alternating steps between optimizing the cluster parameters



(a) Good initialization (b) No residuals (c) Bad global priors

Figure 6.3.: Figure showing the different initializations in latent space. **(a):** an initialization that is likely to converge to a good solution. **(b):** an initial condition where the residual weights are initialized using the normal distribution, and how this results in a different output scale; you can hardly even spot the latent data points. **(c):** sup-optimal priors for the global variables look like; in this case, the clusters are not well spread out and overlap each other, most likely this will converge to a solution where a single cluster is assigned to all the data points.

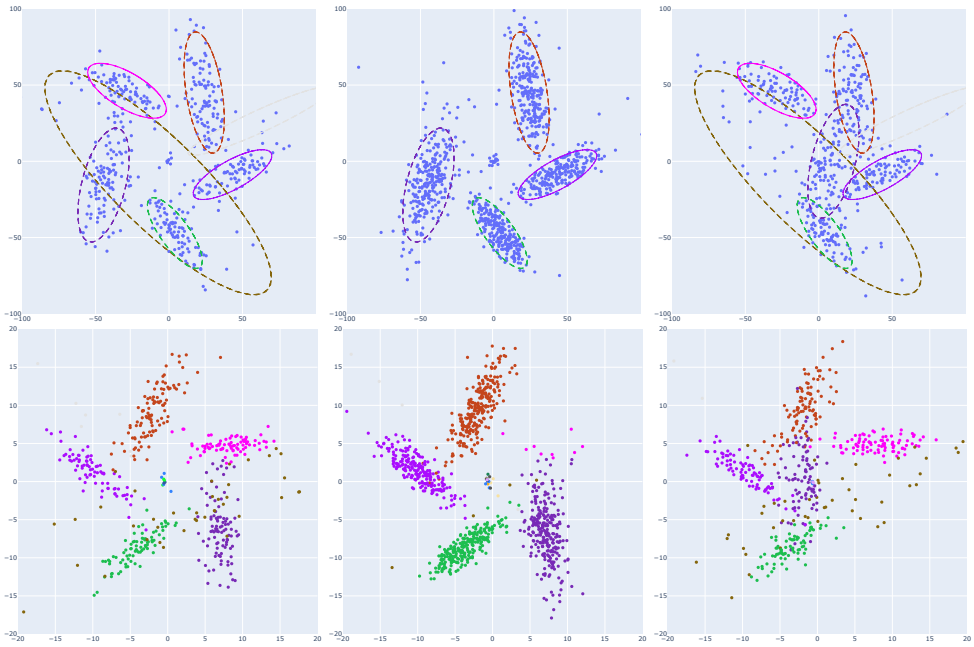
(μ, Σ) and the cluster assignments z. Local optimization depends on the potentials ψ, the output of the encoder. If any of these quantities are initialized poorly, the whole optimization will go in the wrong direction. Bad potentials result in the local variables essentially fitting the wrong observations. Bad cluster parameters result in wrong cluster assignments, and wrong cluster assignments result in bad cluster parameter updates. This spirals quickly out of control.

To combat this, residual connections are used. Residual connections are linear layers going directly from input to output[1]. These connections are initialized using *"random partial isometry"*[2]. This initialization achieves that the output has a similar "scale" as the input. The encoder and decoder weights are initialized close to zero, i.e. their initial outputs are close to zero. As a result, the combined initial outputs are dominated by the residual connections and thus have a similar scale as the inputs due to initialization. Different initialization or no residual connections does not work well (Figure 6.3b).

Additionally, good priors for the global parameters are required. The main importance is that clusters are well distributed over the data points. Otherwise, data points are assigned to distant clusters, and the data will be modeled by a smaller number of clusters than expected. For example, having bad priors resulting in several

[1]The combined output is $\hat{y} = f(x) + Ax$ where $f(x)$ is the neural network and Ax the linear regression.

[2]This initializes the weight matrix using the Q matrix from the QR decomposition. The QR decomposition is taken of the smallest square matrix that encompasses the weight matrix. Each element of this square matrix is normally distributed.



(a) Baseline (b) Remove a cluster (c) Move a cluster

Figure 6.4.: Manipulation of the GMM. The first row shows the latent space; the second row shows the output space. **(a):** shows the baseline. **(b):** shows results for removing a cluster. **(c):** shows the results after moving a cluster.

overlapping clusters (Figure 6.3c) often results in a final model where all data points are assigned to a single, large cluster.

6.1.2. Manipulation of latent variables

In this section, we will show how the GMM can be manipulated. We will show how the latent space and output space change when we move or remove a cluster.

It is possible to remove a cluster by setting the corresponding weight to zero. This will result in the cluster being removed from the latent space, and the corresponding data points will be assigned to the other clusters (see Figure 6.4b).

It is also possible to move a cluster by changing the mean of the corresponding Gaussian. This will result in the cluster being moved into the latent space (see Figure 6.4c).

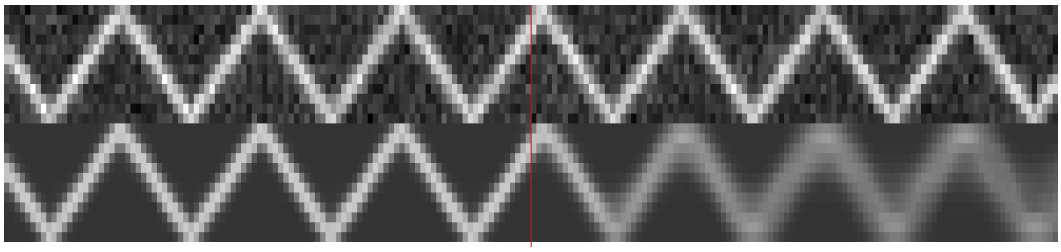


Figure 6.5.: SVAE result for the LDS. The first row shows the training data. The second row shows predictions from the SVAE model, an average of 50 predictions. The red line indicates at which point the potentials are set to zero.

6.2. SVAE-LDS

In the context of this project, synthetic data is generated (see Figure 6.5). Each time step consists of a single frame, which consists of 12 pixels. The goal is then, given some previous frames, to predict the next frame in the sequence.

The SVAE-LDS can model the data well and can extrapolate over a longer period (see Figure 6.5).

6.2.1. Importance of different latent variables

One of the supposed advantages of the SVAE is that it is more interpretable, but is this the case? Is it possible to identify certain latent states that are related to certain properties in the output? To investigate this, single latent variables are multiplied by a scalar. In this experiment, we multiply one of the latent variables $x^{(i)}$ by a factor for all time steps. Using these modified latent variables as input to the decoder, the data is reconstructed. To measure the quality of the reconstruction the mean squared error is used.

One surprising result was that the mean squared error is hardly affected by multiplying a latent variable by 0 (see Figure 6.6). However, this can be attributed to the high correlation between the variables. When $x^{(j)}$ is generated, it includes information from $x^{(i)}$, and thus setting the latter to 0 does not remove the information in the former. Furthermore, the decoder likely has redundancy and does not rely on a single latent input. The second observation is that only for a factor of 100 there is a large difference. This also can be explained by correlation, but also that the decoder is quite robust to manipulations to its input.

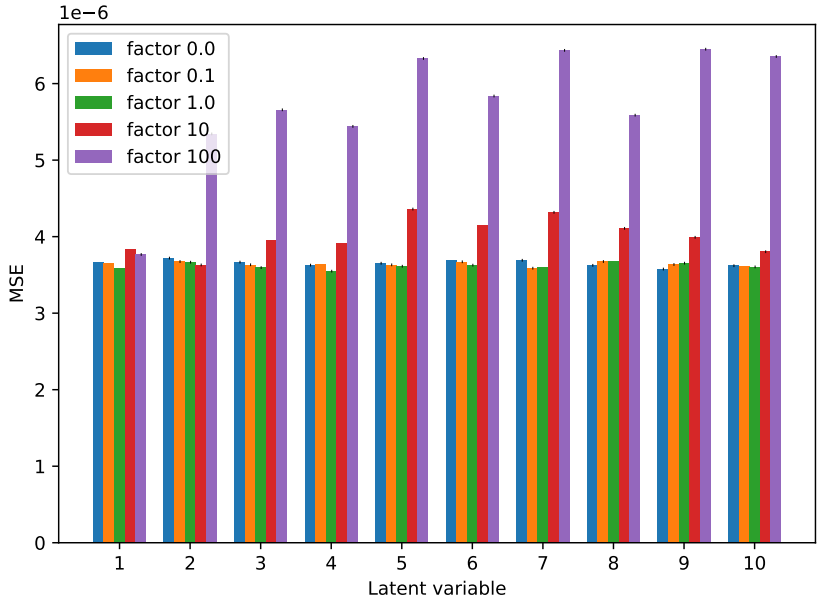


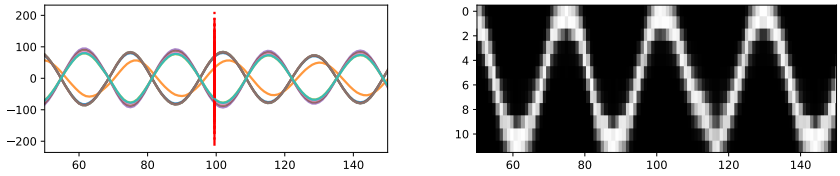
Figure 6.6.: Manipulation of local variables by multiplicative factors. The values shown are the average over 10 runs.

6.2.2. Manipulation of latent variables

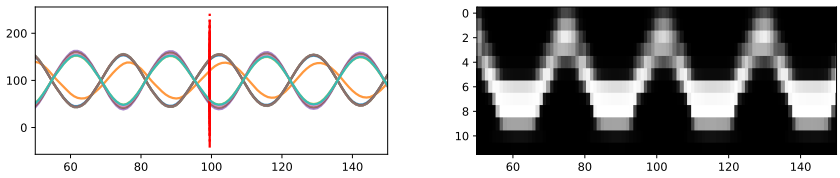
There are more possible ways to manipulate the local latent variables. In the last section, we multiplied latent variables with factors. In this section, we will shift the latent variables and change the input data. The goal is to better understand the relation between the local variables and the reconstruction.

First, to test the robustness of the reconstruction we shift the latent variables by a constant. This is achieved by setting $x^{(i)} = x^{(i)} + \delta$ for all time steps. In this experiment, δ is chosen to be 100, about the same as the range on the y-axis. What this achieves is that the shifted latent variables have no overlap with the non-shifted latent variables. Even though this degrades the reconstruction, it is still recognizable (see Figure 6.7b). This is surprising as the values of the bottom of the shifted latent variables are equal to the top of the non-shifted latent variables. This shows that the decoder is robust to translation invariance, and uses relative differences rather than absolute differences between its inputs.

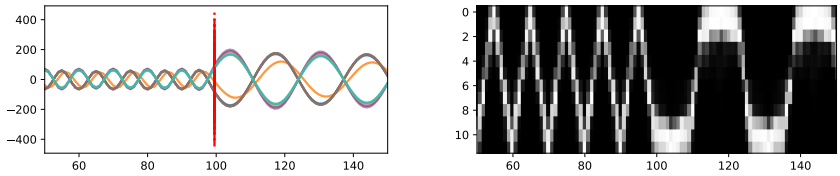
Second, we want to show that the global variables of the PGM indeed learn the properties of the system. This is done by two experiments, in the first one the input frequency of the input data is changed and in the second one, there is no input data at all during evaluation. During training, the input frequency is not changed. In the first experiment, if the SVAE-LDS would use the input data to propagate the input frequency, we would see this frequency continue after feeding the input data.



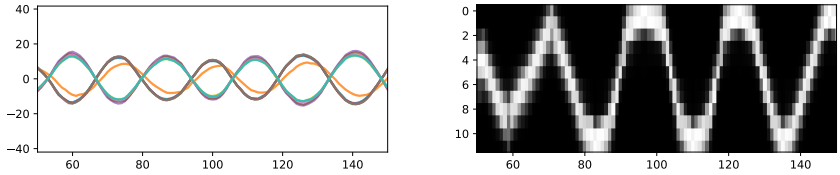
(a) Baseline



(b) Shift y-axis



(c) Frequency of input data



(d) No starting potentials

Figure 6.7.: The left column shows the latent dimension, and the right column shows the reconstruction. Before step 100 the system also receives input data from the current frame to predict the next frame, except for figure (d). **(b):** Shifting the latent variables by a constant. The reconstruction being robust against this shows translation invariance. **(c):** Changing the frequency of the input data after training. This shows that the global parameters learned the correct frequency. **(d):** Not having input data at any point in the evaluation. This shows that the system is not dependent on input after training.

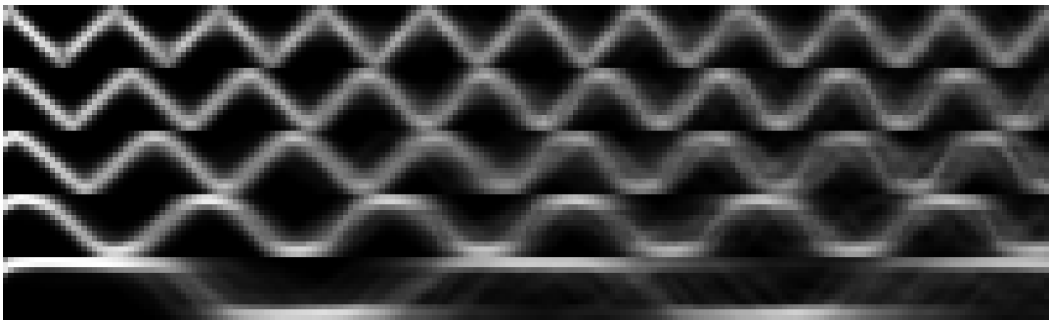


Figure 6.9.: Manipulation of the global variables can change properties of the output. Here a column of the transition matrix is multiplied by factors 0.8, 0.9, 1.0, 1.1, and 1.2, resulting in different output frequencies. In the figure, the average of 50 samples is shown.

This is not the case. Furthermore, in the second experiment, no data at all is fed. The SVAE-LDS is still able to reconstruct the data. This shows that the system models the generative process, implying that we can access properties of the generative process through the global variables.

6.2.3. Manipulation of global variables

In the previous experiments, manipulation was done on local latent variables. However, direct manipulations ignore the dependency between latent variables due to correlation. This correlation occurs in the local optimization of the LDS. Instead, we can manipulate the global transition matrix. This is essentially a step up, and instead of manipulating local variables, the global variables are manipulated.

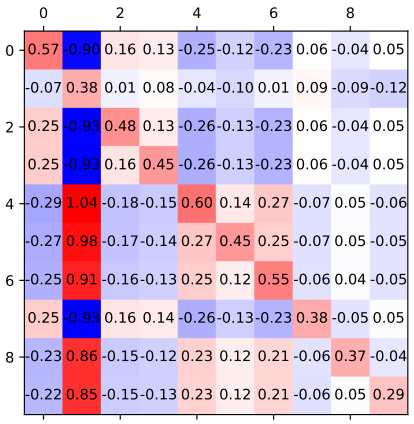
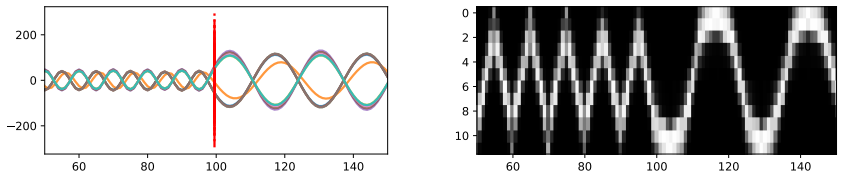
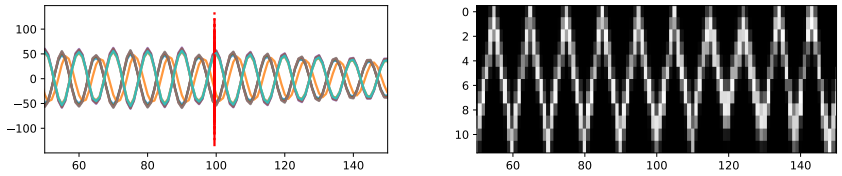


Figure 6.8.: The transition matrix A of the SVAE-LDS.

In this experiment, we manipulate the transition matrix (see Figure 6.8) of a trained SVAE-LDS. This is the matrix A in the relation $x_t = \mathcal{N}(Ax_{t-1}, Q)$. The transition matrix indicates to which extent the value of the latent variable $x_t^{(i)}$ depends on the value of the latent variable $x_{t-1}^{(j)}$, ignoring the noise. Even though there seems to be structure in this matrix, it is difficult to interpret what this means. However, it is possible to manipulate this matrix and see what happens to the output.



(a) Not retrained.



(b) Retrained

Figure 6.10.: Retraining of the global variables can be used to change the learned frequency. The left column shows the latent variables, and the right column shows the output. The frequency of the input data is changed. In the first row, the SVAE-LDS is not retrained and reproduces the frequency of the original data. In the second row, the global variables are updated.

It is difficult to quantify the effect of manipulation. However, in this particular example, we identified a column of the transition matrix that is semi-responsible for the frequency of the output (see Figure 6.9). This column is the 7th column in the transition matrix. Note that this column does not seem special in any way if we just look at the transition matrix.

6.2.4. Retraining of global variables

It is also possible to retrain the global variables and get a different frequency. In the SVAE algorithm, the natural gradient is used to update the global variables (Algorithm 1). First, the potentials ψ are computed by passing the new data through the encoder. This is only done once. Second, the local optimization is done to get the statistics $\hat{t}_x$. Finally, the natural gradient is computed and used to update the global parameters. The decoder is not needed.

By doing the partial updates of the SVAE-LDS it is possible to retrain the global variables (see Figure 6.10). Doing partial updates is much faster than training the whole system. Furthermore in theory it is possible to use Bayesian updating here, and even a single example could be used to update the global variables.

7. Conclusion

The first goal of this thesis was to make the SVAE more accessible. This was achieved by providing a detailed explanation of the background and mathematical details. Furthermore, the algorithm is implemented in PyTorch and made available. Both the SVAE-GMM and the SVAE-LDS performed well on their respective tasks.

The second goal was to investigate the interpretability of the SVAE. For the SVAE-GMM we tested our understanding by manipulating the model. We found that moving or removing a cluster in the latent space has the expected result in the reconstruction. While it is possible to do similar manipulations for a neural network, it lacks the interpretability of the SVAE-GMM.

More evidence for the interpretability of the SVAE was found in experiments done on the SVAE-LDS. Using manipulations of the local and global variables we found that the SVAE-LDS shows translation invariance and that the global parameters learn the overall dynamics of the data. The latter was shown by the model's ability to generate the data during testing, without any access to the data. Because of the structure of the SVAE-LDS it is possible to conclude from that that the global parameters learned the dynamics. This is also empirically supported as it was shown that it is possible to change the frequency of the output by manipulating global parameters. This change can also be systematically induced, by retraining the global parameters on new data.

However, the SVAE also has its share of problems. In the case of the SVAE-GMM we saw that the initial conditions are important. If they are not chosen well, the SVAE will not produce a good model. While we can see this in low-dimensional data, choosing good initial conditions is difficult in high-dimensional data. This is a problem inherited from the GMM, and is not specific to the SVAE.

This is not an issue for the SVAE-LDS, but experiments showed another problem. We found that the local latent variables of the LDS were highly correlated. This is a behavior often seen within neural networks, and the SVAE also suffers from this. Due to the high correlation, it becomes difficult to interpret the SVAE. While it's true that we have access to the parameters of the graphical model, it can still be difficult to interpret these parameters.

We hope that this work will inspire others to explore the possibilities of combining neural networks and graphical models. This thesis and the accompanying code provide a good starting point for this exploration. The empirical results for the interpretability of the SVAE are promising. However, more work is needed to fully

understand if the composition of neural networks and graphical models is a solution to the interpretability issue of neural networks.

A. Mixture model updates

A.1. Optimizing mean-field factor

The derivations here are partly based on contents of the appendix of [JDW+16].

Proposition 3. *Assume that we have the following mean field variational inference objective*

$$\mathop{\mathbb{E}}_{q(a)q(b)q(c)}\Big[\log \frac{p(a,b,c)}{q(a)q(b)q(c)}\Big] \tag{A.1}$$

For fixed $q(a), q(c)$ the partially optimal factor $q^*(b)$

$$q^*(b) \triangleq \arg\max_{q(b)} \mathop{\mathbb{E}}_{q(a)q(b)q(c)}\left[\log \frac{p(a,b,c)}{q(a)q(b)q(c)}\right], \tag{A.2}$$

is defined by

$$q^*(b) \propto \exp\{\mathop{\mathbb{E}}_{q(a)q(c)} \log p(a,b,c)\} \tag{A.3}$$

In particular if $p(c|b,a)$ is an exponential family with $p(b|a)$ its natural exponential family conjugate prior and $\log p(b,c|a)$ is a multi-linear polynomial in the statistics $t_b(b)$ and $t_c(c)$ written

$$p(b|a) = \exp\{\langle \eta_b^0(a), t_b(b)\rangle - \log Z_b(\eta_b^0(a))\} \tag{A.4}$$

$$\begin{aligned} p(c|b,a) &= \exp\{\langle \eta_c^0(b,a), t_c(c)\rangle - \log Z_c(\eta_c^0(b,a))\} \\ &= \exp\{\langle t_b(b), \eta_c^0(a)^T (t_c(c), 1)\rangle\} \end{aligned} \tag{A.5}$$

for some matrix $\eta_c^0(a)$. Then the optimal factor can be written as

$$q^*(b) = \exp\{\langle \eta_b^*, t_b(b)\rangle - \log Z_b(\eta_b^*)\} \tag{A.6}$$

with

$$\eta_b^* \triangleq \mathop{\mathbb{E}}_{q(a)}\Big[\eta_b^0(a)\Big] + \mathop{\mathbb{E}}_{q(a)q(c)}\Big[\langle \eta_c^0(a), (t_c(c), 1)\rangle\Big] \tag{A.7}$$

If $c \perp\!\!\!\perp b|a$ then

$$\eta_b^* \triangleq \mathop{\mathbb{E}}_{q(a)}\Big[\eta_b^0(a)\Big] + \mathop{\mathbb{E}}_{q(c)}\Big[(t_c(c), 1)\Big] \tag{A.8}$$

Proof. Start by rewriting the expectation in (A.2)

$$\begin{aligned}\mathop{\mathbb{E}}_{q(a)q(b)q(c)}\left[\log\frac{p(a,b,c)}{q(a)q(b)q(c)}\right] &= \mathop{\mathbb{E}}_{q(b)}\left[\mathop{\mathbb{E}}_{q(a)q(c)}\log p(a,b,c) - \log q(b)\right] \\ &= \mathop{\mathbb{E}}_{q(b)}\left[\log\exp\mathop{\mathbb{E}}_{q(a)q(c)}\log p(a,b,c) - \log q(b)\right] \\ &= -\mathop{\mathbb{E}}_{q(b)}\left[\frac{q(b)}{\tilde{p}(b)}\right] + \text{const} \\ &= -\mathbb{KL}(q(b)||\tilde{p}(b)) + \text{const}\end{aligned} \tag{A.9}$$

which is maximized by setting $q(b)$ equal to $\tilde{p}(b)$ as then $\mathbb{KL}$ is zero. The rest follows by plugging in exponential family densities. Note the similarity of this derivation and sections 2.1, 2.2. □

A.2. Additional conjugacy structure of the local latent variables

We can assume additional structure in the local latent variables. This allows us to exploit this structure and make inference faster. We do this by introducing an additional set of local latent variables z, in addition to the local latent variables x. I.e. here we are adding local latent variable z to the model described at the start of this section.

Let $p(z,x|\theta)$ be an exponential family and $p(\theta)$ its corresponding natural exponential family conjugate prior,

$$p(\theta) = \exp\{\langle\eta_\theta^0, t_\theta(\theta)\rangle - \log Z_\theta(\eta_\theta^0)\} \tag{A.10}$$

$$\begin{aligned}p(z,x|\theta) &= \exp\{\langle\eta_{zx}^0(\theta), t_{zx}(z,x)\rangle - \log Z_{zx}(\eta_{zx}^0(\theta))\} \\ &= \exp\{\langle t_\theta(\theta), (t_{zx}(z,x),1)\rangle\}\end{aligned} \tag{A.11}$$

Let $t_{zx}(z,x)$ be a multi-linear polynomial in the statistics $t_z(z)$ and $t_x(x)$, and let $p(z|\theta)$ and $p(x|z,\theta)$ be a conjugate pair of exponential families, writing

$$p(z|\theta) = \exp\{\langle\eta_z^0(\theta), t_z(z)\rangle - \log Z_z(\eta_z^0(\theta))\} \tag{A.12}$$

$$\begin{aligned}p(x|z,\theta) &= \exp\{\langle\eta_x^0(z,\theta), t_x(x)\rangle - \log Z_z(\eta_x^0(z,\theta))\} \\ &= \exp\{\langle t_z(z), \eta_x^0(\theta)^T(t_x(x),1)\rangle\}\end{aligned} \tag{A.13}$$

Let $p(y|x,\gamma)$ be a general family of densities and let $p(\gamma)$ be an exponential family prior on its parameters of the form

$$p(\gamma) = \exp\{\langle\eta_\gamma^0, t_\gamma(\gamma)\rangle - \log Z_\gamma(\eta_\gamma^0)\} \tag{A.14}$$

As before, construct a mean field objective

$$\mathcal{L}(\eta_\theta, \eta_\gamma, \eta_z, \eta_x) \triangleq \mathop{\mathbb{E}}_{q(\theta)q(\gamma)q(z)q(x)}\left[\log \frac{p(\theta)p(\gamma)p(z|\theta)p(x|z,\theta)p(y|x,\gamma)}{q(\theta)q(\gamma)q(z)q(x)}\right] \tag{A.15}$$

with the surogate objective

$$\hat{\mathcal{L}}(\eta_\theta, \eta_\gamma, \eta_x, \phi) \triangleq \mathop{\mathbb{E}}_{q(\theta)q(z)q(x)}\left[\log \frac{p(\theta)p(z|\theta)p(x|z,\theta)\psi(x;y,\phi)}{q(\theta)q(z)q(x)}\right] \tag{A.16}$$

$$\mathcal{L}_{\text{SVAE}}(\eta_\theta, \eta_\gamma, \phi) \triangleq \mathcal{L}(\eta_\theta, \eta_\gamma, \eta_z^*(\eta_\theta, \phi), \eta_x^*(\eta_\phi)) \tag{A.17}$$

Where $\eta_z^*(\eta_\theta, \phi)$ and $\eta_x^*(\eta_\phi)$ are the local partial optimizers of $\hat{\mathcal{L}}$ given fixed values of the other parameters η_θ, ϕ and satisfy the first order optimality conditions

$$\nabla_{\eta_z} \hat{\mathcal{L}}(\eta_\theta, \eta_z^*(\eta_\theta, \phi), \eta_z^*(\eta_\theta, \phi), \phi) = 0 \tag{A.18}$$

$$\nabla_{\eta_x} \hat{\mathcal{L}}(\eta_\theta, \eta_z^*(\eta_\theta, \phi), \eta_z^*(\eta_\theta, \phi), \phi) = 0 \tag{A.19}$$

Then we get that η_z^* and η_x^* are given by

$$\eta_z^* = \mathop{\mathbb{E}}_{q(\theta)}\left[\eta_z^0(\theta)\right] + \mathop{\mathbb{E}}_{q(\theta)q(x)}\left[\langle \eta_x^0(\theta), (t_x(x), 1)\rangle\right] \tag{A.20}$$

$$\eta_x^* = \mathop{\mathbb{E}}_{q(\theta)q(z)}\left[\eta_x^0(z, \theta)\right] + r(y;\phi) \tag{A.21}$$

The updates follow from the previous section A.1.

A.3. Code

Next, we will take a look at the local optimization for the GMM. These updates are derived in A.2. Especially see Equation (A.21) for the updates shown in `gaussian_optimization`, and Equation (A.20) for the updates shown in `label_optimization` .

```
def gaussian_optimization(gaussian_parameters, potentials, label_stats):
  # message from parent to child
  gaussian_potentials = torch.tensordot(
      label_stats, gaussian_parameters, [[1], [0]]
  )
  # update parameters
  # message from parent + message from child
  eta_x = gaussian_potentials + potentials
  # message from child to parent
  gaussian_stats = Gaussian(eta_x).expected_stats()
  gaussian_kld = (
      torch.tensordot(potentials, gaussian_stats, 3) - Gaussian(eta_x).
                                                       logZ()
  )
  return eta_x, gaussian_stats, gaussian_kld.item()
```

```
def label_optimization(gaussian_parameters, label_parameters,
                                        gaussian_stats):
  # message from parent to child
  label_potentials = torch.tensordot(
      gaussian_stats, label_parameters, [[1], [0]]
  )
  # update parameters
  # message from parent + message from child
  eta_z = label_potentials + potentials
  # message from child to parent
  label_stats = Dirichlet(eta_z).expected_stats()
  label_kld = (
      torch.tensordot(potentials, label_stats, 3) - Dirichlet(eta_z).logZ
                                                    ()
  )
  return eta_z, label_stats, label_kld.item()
```

B. Information form operations

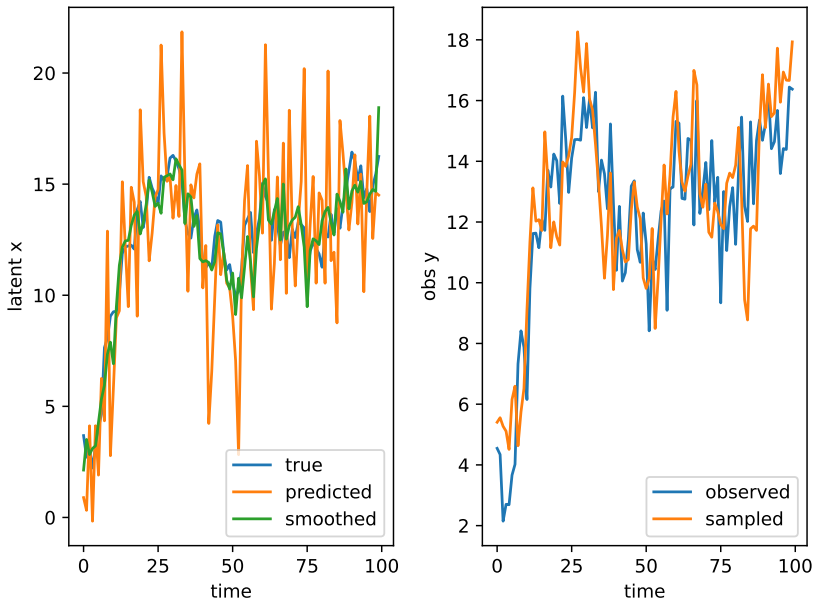


Figure B.1.: Example of a Linear Dynamical System. **Left:** shows the true, predicted and smoothed latent variables. **Right:** shows the true and predicted observations.

One way to parametrize the Multivariate Gaussian is the *Information Form* (B.1)[1]. This particular form makes the operations required for Kalman filtering and smoothing much easier.

[1]Appendix D.4 shows the Standard and Exponential forms

Information Form

$$p(x|J,h) = \exp\left\{-\frac{1}{2}x^T Jx + h^T x - \log Z\right\} \tag{B.1}$$

where

$$\log Z = \frac{1}{2}h^T J^{-1}h - \frac{1}{2}\log|J| + \frac{n}{2}\log 2\pi$$

and

$$\Sigma = J^{-1}$$

$$\mu = J^{-1}h$$

To do Kalman filtering and smoothing there are two important operations, *conditioning* and *marginalization*. These operations are defined using the information form of the Gaussian, as this results in an easier conditioning step.

Conditioning

If,

$$p(x) = \mathcal{N}(x|J,h) \tag{B.2}$$

$$p(y|x) \propto \mathcal{N}(x|J_{\text{obs}},h_{\text{obs}}) \tag{B.3}$$

then,

$$p(x|y) = \mathcal{N}(x|J + J_{\text{obs}},h + h_{\text{obs}}) \tag{B.4}$$

Marginalization

If,

$$\begin{bmatrix}x_1\\x_2\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}J_{11} & J_{12}\\J_{12}^T & J_{22}\end{bmatrix},\begin{bmatrix}h_1\\h_2\end{bmatrix}\right) \tag{B.5}$$

then,

$$x_2 \sim \mathcal{N}(J_{22} - J_{12}^T J_{11}^{-1}J_{12}, h_2 - J_{12}^T J_{11}^{-1}h_1) \tag{B.6}$$

with the normalization constant,

$$\log Z - \frac{1}{2}h_1^T J_{11}^{-1}h_1 + \frac{1}{2}\log|J_{11}| - \frac{n}{2}\log 2\pi \tag{B.7}$$

B.1. Filtering, Sampling, and Smoothing

Using the information form, with corresponding conditioning and marginalization steps, *filtering*, *sampling*, and *smoothing* are done as follows. Take the following model

$$x_1 \sim \mathcal{N}(\mu_1, Q_1) \tag{B.8}$$
$$x_{t+1} \sim \mathcal{N}(A_t x_t + B_t u_t, Q_t) \tag{B.9}$$
$$y_t \sim \mathcal{N}(C_t x_t + D_t u_t, R_t) \tag{B.10}$$

Writing this in the information form we get that the initial distribution is

$$x_1 \sim \mathcal{N}(J = Q_1^{-1}, h = Q^{-1}\mu_1) \tag{B.11}$$

The transition dynamics are given by

$$p(x_{t+1}|x_t) \propto \mathcal{N}\left(\begin{bmatrix} J_{11} & J_{12} \\ J_{12}^T & J_{22} \end{bmatrix}, \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}\right) \tag{B.12}$$

with

$$J_{11} = A_t^T Q_t^{-1} A_t \tag{B.13}$$
$$J_{12} = -A_t^T Q_t^{-1} \tag{B.14}$$
$$J_{11} = Q_t^{-1} \tag{B.15}$$
$$h_1 = -u_t^T B_t^T A_t \tag{B.16}$$
$$h_2 = u_t^T B_t Q_t^{-1} \tag{B.17}$$

and the observations are given by

$$p(y_t|x_t) \propto \mathcal{N}(x_t|J_{\text{obs}}, h_{\text{obs}}) \tag{B.18}$$
$$J_{\text{obs}} = C_t^T R_t^{-1} C_t \tag{B.19}$$
$$h_{\text{obs}} = (y_t - D_t u_t) R_t^{-1} C_t \tag{B.20}$$

B.2. Filtering

Begin with the initial distribution

$$p(x_1) = \mathcal{N}(x_1|J_{1|0}, h_{1|0}). \tag{B.21}$$

Assume, inductively, that $x_t|y_{1:t-1} \sim \mathcal{N}(J_{t|t-1}, h_{t|t-1})$. Conditioning on the t-th observation yields

$$p(x_t|y_{1:t}) = \mathcal{N}(x_t|J_{t|t}, h_{t|t}) \tag{B.22}$$
$$J_{t|t} = J_{t|t-1} + J_{\text{obs}} \tag{B.23}$$
$$h_{t|t} = h_{t|t-1} + h_{\text{obs}} \tag{B.24}$$

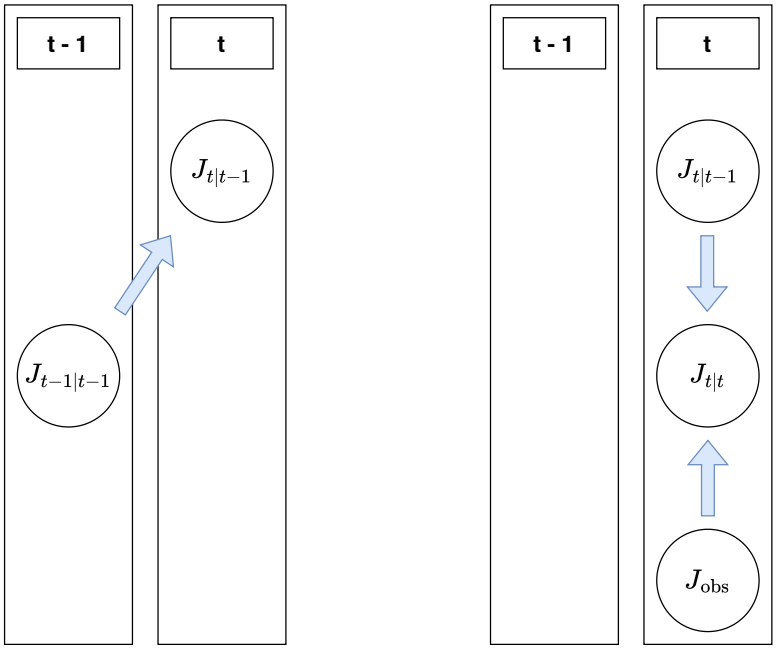


Figure B.2.: Forward Kalman filter. Shown are the steps for J, but they are analogous for h. **Left:** shows the prediction step (B.26). **Right:** shows the conditioning step (B.23). In the context of the SVAE J_{obs} are the potentials (section 3). In order to estimate the current parameters: First, predict the estimation $J_{t|t-1}$ using the previous estimation $J_{t-1|t-1}$; Second, combine $J_{t|t-1}$ with the actual observation J_{obs} to obtain $J_{t|t}$.

Then predict the next latent state by writing the joint distribution of x_t and x_{t+1}, and marginalize out x_t. This completes one iteration, and provides input to the next.

$$p(x_{t+1}|y_{1:t}) = p(x_t|y_{1:t})p(x_{t+1}|x_t) = \mathcal{N}(x_t|J_{t+1|t}, h_{t+1|t}) \tag{B.25}$$

$$J_{t+1|t} = J_{22} - J_{12}^T \left(J_{t|t} + J_{11}\right)^{-1} J_{12} \tag{B.26}$$

$$h_{t+1|t} = h_2 - J_{12}^T \left(J_{t|t} + J_{11}\right)^{-1} \left(h_{t|t} - h_1\right) \tag{B.27}$$

In the code the Python implementation of the forward filter is shown (Figure B.2). The main function is `info_kalman_filter`, which repeats the condition and predict operations for every time step t. For every time step t: do the condition step in `info_condition`, then do the predict step in `info_predict`. Without the information parameterization the function `info_condition` would be much more verbose. The `info_predict` function is just `info_marginalize` with an extra step.

```
def info_condition(J, h, J_obs, h_obs):
  return J + J_obs, h + h_obs

def info_marginalize(J11, J12, J22, h):
  J_pred = J22 - J12.T @ torch.linalg.solve(J11, J12)
  h_pred = -J12.T @ torch.linalg.solve(J11, h)
  return J_pred, h_pred

def info_predict(J, h, J11, J12, J22):
  J_new = J + J11
  return info_marginalize(J_new, J12, J22, h)

def info_kalman_filter(init_params, pair_params, observations):
  J, h = init_params
  J11, J12, J22 = pair_params

  forward_messages = []
  for (J_obs, h_obs) in observations:
    J_cond, h_cond = info_condition(J, h, J_obs, h_obs)
    J, h = info_predict(J_cond, h_cond, J11, J12, J22)
    forward_messages.append(((J_cond, h_cond), (J, h)))

  return forward_messages
```

B.3. Rauch-Tung-Striebel Smoothing

To obtain the natural gradient for the global parameters the expected statistics are required (section 3.2). In turn, these expected statistics require the conditional distribution given all the data. This is done using smoothing.

$$\begin{aligned} J_{t|T} &= J_{t|t} + J_{11} - J_{12}(J_{t+1|T} - J_{t+1|t} + J_{22})^{-1} J_{12}^T \\ h_{t|T} &= h_{t|t} + h_1 - J_{12}(J_{t+1|T} - J_{t+1|t} + J_{22})^{-1}(h_{t+1|T} - h_{t+1|t} + h_2) \end{aligned} \tag{B.28}$$

Smoothing is implemented across two different Python functions. The main function `info_kalman_smoothing` takes care of the loop going back through time and saving the smoothed parameters. The helper function `info_rts_smoothing` contains the update equations (B.28) doing the smoothing.

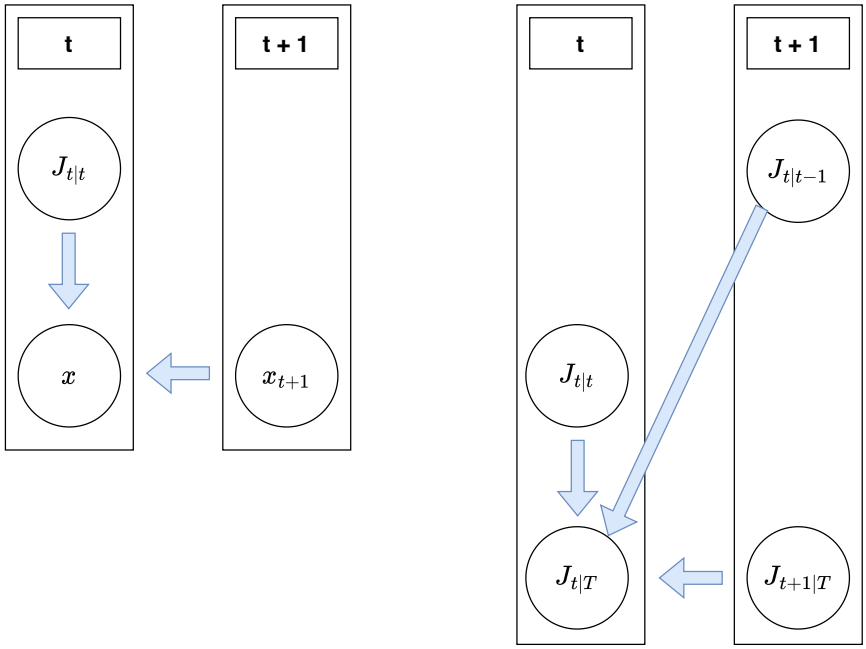


Figure B.3.: Kalman backward sampling and smoothing, both going back through time. **Left:** obtain current sample x_t using estimated parameters $J_{t|t}$ from the filtering operation and future sample x_{t+1}. **Rigth:** smooth the current estimated parameters $J_{t|t}$ into $J_{t|T}$ by using information from the future. This future information is available from the previous filtering operation.

```
def info_kalman_smoothing(forward_messages, pair_params):
  (J_smooth, h_smooth), _ = forward_messages[-1]
  loc, scale = info_to_standard(J_smooth, h_smooth)
  E_xxT = scale + outer_product(loc, loc)
  E_xnxT = 0.0

  expected_stats = [(loc, E_xxT, E_xnxT)]
  backward_messages = [(J_smooth, h_smooth)]
  for i, (cond_msg, pred_msg) in enumerate(reversed(forward_messages[:-1]
                                                    )):
      loc_next, _, _ = expected_stats[i]
      J_smooth, h_smooth, stats = info_rst_smoothing(
          J_smooth, h_smooth, cond_msg, pred_msg, pair_params, loc_next
      )
      backward_messages.append((J_smooth, h_smooth))
      expected_stats.append(stats)

  expected_stats = process_expected_stats(list(reversed(expected_stats)))
  return list(reversed(backward_messages)), expected_stats
```

```
def info_rts_smoothing(J, h, cond_msg, pred_msg, pair_params, loc_next):
  J_cond, h_cond = cond_msg
  J_pred, h_pred = pred_msg
  J11, J12, J22 = pair_params

  temp = J - J_pred + J22
  J_smooth, h_smooth = info_marginalize(
      A=temp, B=J12, C=(J_cond + J11), h1=h - h_pred, h2=h_cond)

  loc, scale = info_to_standard(J_smooth, h_smooth)
  E_xnxT = -torch.linalg.solve(temp, J12.T) @ scale + outer_product(
                                        loc_next, loc)
  E_xxT = scale + outer_product(loc, loc)
  stats = (loc, E_xxT, E_xnxT)

  return J_smooth, h_smooth, stats
```

B.4. Backward sampling

After computing good estimates for $J_{t|t}$ and $h_{t|t}$, sampling can be done by going back in time. Using the conditional for x_t, draw the first sample. Then, using this sample, draw the next sample x_{t-1}. Repeat this step until the first time step is reached, and the last sample x_1 is obtained.

$$\begin{aligned} p(x_t|y_{1:t}, x_{t+1}) &\propto p(x_t|y_{1:T})p(x_{t+1}|x_t)) \\ &\propto \mathcal{N}(x_t|J_{t|t}, h_{t|t})\mathcal{N}(x_t|J_{11}, h_1 - x_{t+1}^T J_{12}^T) \\ &\propto \mathcal{N}(x_t|J_{t|t} + J_{11}, h_{t|t} + h_1 - x_{t+1}J_{12}^T) \end{aligned} \tag{B.29}$$

In the following code segment, the Python implementation of backward sampling is shown. As described above, the first sample is obtained by using the estimated parameters for the last time step. Then, until the first time step is reached, use the previous sample (one step in the future) to obtain the next sample (current time step).

```
def info_sample_backward(forward_messages, pair_params):
  J11, J12, J22 = pair_params

  J_pred, h_pred = forward_messages[-1]
  next_sample = Gaussian(J_pred, h_pred).rsample()

  samples = [next_sample]
  for J_pred, h_pred in reversed(forward_messages[:-1]):

    J = J_pred + J11
    h = h_pred - next_sample @ J12

    # get the sample
    next_sample = Gaussian(J, h).rsample()
    samples.append(next_sample)

  return list(reversed(samples))
```

C. Exponential family

C.1. Definition

Given a *statistic function* $t_x : \mathcal{X} \to \mathbb{R}^n$ and a a *base measure* $\nu_{\mathcal{X}}$, we can define an exponential family of probability densities on $\mathcal{X}$ relative to $\nu_{\mathcal{X}}$ and indexed by the *natural parameter* $\eta_x \in \mathbb{R}^n$ by

$$p(x|\eta_x) \propto \exp\{\eta_x^T t_x(x)\}, \quad \forall \eta_x \in \mathbb{R}^n \tag{C.1}$$

i.e. for every specific η_x we have one member of the family. In case we want equality, rather then proportionality, we need a *normalization constant*. This is defined as follows

$$Z_x(\eta_x) \triangleq \int \exp\{\eta_x^T t_x(x)\} \nu_{\mathcal{X}}(dx), \tag{C.2}$$

giving us the equality

$$p(x|\eta_x) = \exp\{\eta_x^T t_x(x) - \log Z_x(\eta_x)\}. \tag{C.3}$$

Additionaly we can introduce parameter θ, and parametrize the natural parameter with θ

$$p(x|\theta) = \exp\{\eta_x(\theta)^T t_x(x) - \log Z_x(\eta_x(\theta))\}, \tag{C.4}$$

Definition C.1.1. *We define the natural parameter space as the set of all normalizable natural parameters*

$$H \triangleq \{\eta \in \mathbb{R}^n | Z_x(\eta) < \infty\}, \tag{C.5}$$

By $\Theta = \eta_x^{-1}(H)$ *we denote the set of parameters that correspond to normalizable densities.*

C.2. Conjugacy

Given the definition of the exponential family $p(x|\theta)$ as in (C.4), define the statistic function $t_\theta : \Theta \to \mathbb{R}^{n+1}$ as the concatenation

$$t_\theta(\theta) \triangleq (\eta_x(\theta), -\log Z_x(\eta_x(\theta))). \tag{C.6}$$

We call the exponential family with statistic function $t_\theta(\theta)$, denoted by

$$p(\theta) = \exp\{\eta_\theta^T t_\theta(\theta) - \log Z_\theta(\eta_\theta)\} \tag{C.7}$$

the *conjugate prior* of $p(x|\theta)$. We can use the prior statistic function $t_\theta(\theta)$ to rewrite the likelihood $p(x|\theta)$

$$\begin{aligned} p(x|\theta) &= \exp\{\eta_x(\theta)^T t_x(x) - \log Z_x(\eta_x(\theta))\} \\ &= \exp\{t_\theta(\theta)^T (t_x(x), 1)\}. \end{aligned} \tag{C.8}$$

We will now look at the conjugacy properties of (C.4) and (C.7). First all we have

$$\begin{aligned} p(x,\theta) &= p(x|\theta)p(\theta) \\ &= \exp\{t_\theta(\theta)^T (t_x(x),1)\} \exp\{\eta_\theta^T t_\theta(\theta) - \log Z_\theta(\eta_\theta)\} \\ &= \exp\{(t_x(x),1)^T t_\theta(\theta)\} \exp\{\eta_\theta^T t_\theta(\theta) - \log Z_\theta(\eta_\theta)\} \\ &= \exp\{((t_x(x),1) + \eta_\theta)^T t_\theta(\theta) - \log Z_\theta(\eta_\theta)\}. \end{aligned} \tag{C.9}$$

Second, we have

$$\begin{aligned} p(\theta|x) &= p(x,\theta)/p(x) \\ &= \exp\{((t_x(x),1) + \eta_\theta)^T t_\theta(\theta) - \log Z_\theta((t_x(x),1) + \eta_\theta)\}. \end{aligned} \tag{C.10}$$

The above equations show that the posterior $p(\theta|x)$ with the natural parameter $\eta_\theta + (t_x(x), 1)$ is in the same exponential family as $p(\theta)$.

C.3. $\mathbb{KL}$-Divergence

The Kullback-Leibler divergence ($\mathbb{KL}$) is defined as

$$\mathbb{KL}(p(x;\eta_1)||p(x;\eta_2)) = \int p(x;\eta_1) \log \frac{p(x;\eta_1)}{p(x;\eta_2)} dx \tag{C.11}$$

$$= \mathop{\mathbb{E}}_{p(x;\eta_1)} \left[\log \frac{p(x;\eta_1)}{p(x;\eta_2)} \right] \tag{C.12}$$

Then if the distributions are exponential family we get that

$$\mathbb{KL}(p(x;\eta_1)||p(x;\eta_2)) = \mathop{\mathbb{E}}_{p(x;\eta_1)} \left[\log \frac{p(x;\eta_1)}{p(x;\eta_2)} \right] \tag{C.13}$$

$$= \mathop{\mathbb{E}}_{p(x;\eta_1)} \left[\log \frac{\exp(\eta_1^T t(x) - \log Z(\eta_1))}{\exp(\eta_2^T t(x) - \log Z(\eta_2))} \right] \tag{C.14}$$

$$= \mathop{\mathbb{E}}_{p(x;\eta_1)} \left[(\eta_1 - \eta_2)^T t(x) - \log Z(\eta_1) + \log Z(\eta_2) \right] \tag{C.15}$$

$$= (\eta_1 - \eta_2)^T \mathop{\mathbb{E}}_{p(x;\eta_1)} [t(x)] - \log Z(\eta_1) + \log Z(\eta_2) \tag{C.16}$$

$$= (\eta_1 - \eta_2)^T \mu_1 - \log Z(\eta_1) + \log Z(\eta_2) \tag{C.17}$$

$$\tag{C.18}$$

C.4. Expected Value

For the exponential family we have that

$$\int h(x)\exp\{\eta^T T(x) - A(\eta)\}dx = 1 \tag{C.19}$$

which implies that

$$A(\eta) = \log \int h(x)\exp\{\eta^T T(x)\}dx \tag{C.20}$$

The first derivative of $A(\eta)$ is the expected value, as shown below

$$\begin{aligned}
\frac{\delta A(\eta)}{\delta \eta} &= \frac{\delta}{\delta \eta}\left(\log \int h(x)\exp\{\eta^T T(x)\}dx\right) \\
&= \frac{\int h(x)\exp\{\eta^T T(x)\}T(x)dx}{\exp A(\eta)} \\
&= \int h(x)\exp\{\eta^T T(x) - A(\eta)\}T(x)dx \\
&= \int p(x|\eta)T(x)dx \\
&= \mathop{\mathbb{E}}_{\eta}[T(x)]
\end{aligned} \tag{C.21}$$

D. Distributions

D.1. Dirichlet

Standard Form

$$p(x|\alpha) = \frac{1}{\beta(\alpha)} \prod_{i=1}^{k} x_i^{\alpha_i - 1} \tag{D.1}$$

with the beta function

$$\beta(\alpha) = \frac{\prod_{i=1}^{k} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{k} \alpha_i)} \tag{D.2}$$

and standard parameter α.

Exponential Form

$$p(x|\alpha) = \exp\{\langle \alpha - 1, \log x \rangle - \log \beta(\alpha)\} \tag{D.3}$$

with natural parameter $\eta_1 = \alpha - 1$ and corresponding statistic $t_1 = \log x$.

The following gives use the exponential form:

$$\begin{aligned}
\exp \log p(x|\alpha) &= \exp\{\log \frac{1}{\beta(\alpha)} \prod_{i=1}^{k} x_i^{\alpha_i - 1}\} \\
&= \exp\{\log \prod_{i=1}^{k} x_i^{\alpha_i - 1} - \log \beta(\alpha)\} \\
&= \exp\{\sum_{i=1}^{k} (\alpha_i - 1) \log x_i - \log \beta(\alpha)\} \\
&= \exp\{\langle \alpha - 1, \log x \rangle - \log \beta(\alpha)\}
\end{aligned}$$

Expected Statistics

$$\mathbb{E}[\log x] = \psi(\alpha_j) - \psi(\sum_{i=1}^{k} \alpha_i) \tag{D.4}$$

The following gives the derivative of the log normalization constant:

$$\begin{aligned}\frac{\partial \log Z}{\partial \alpha_j} &= \frac{\partial \log \beta(\alpha_j)}{\partial \alpha} \\ &= \frac{\partial}{\partial \alpha_j}\left[\log \frac{\prod_{i=1}^{k}\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{k}\alpha_i)}\right] \\ &= \sum_{i=1}^{k}\frac{\partial}{\partial \alpha_j}\left[\log \frac{\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{k}\alpha_i)}\right] \\ &= \sum_{i=1}^{k}\frac{\partial}{\partial \alpha_j}\left[\log\Gamma(\alpha_i) - \log\Gamma(\sum_{i=1}^{k}\alpha_i)\right] \\ &= \psi(\alpha_j) - \psi(\sum_{i=1}^{k}\alpha_i)\end{aligned}$$

D.2. Categorical

Standard Form

$$f(x|p) = \prod_{i=1}^{k} p_i^{x_i} \tag{D.5}$$

and standard parameter p.

Exponential Form

$$f(x|p) = \exp\{\langle x, \log p\rangle\} \tag{D.6}$$

with natural parameter $\eta_1 = \log p$ and corresponding statistic $t_1 = x$.

The following gives use the exponential form:

$$\begin{aligned}\exp\log f(x|p) &= \exp\{\log \prod_{i=1}^{k} p_i^{x_i}\} \\ &= \exp\{\sum_{i=1}^{k} x_i \log p_i\} \\ &= \exp\{\langle x, \log p\rangle\}\end{aligned}$$

Expected Statistics

$$\mathbb{E}[\log x] = \psi(\alpha_j) - \psi(\sum_{i=1}^{k} \alpha_i) \tag{D.7}$$

The following gives the derivative of the log normalization constant:

$$\begin{aligned}
\frac{\partial \log Z}{\partial \alpha_j} &= \frac{\partial \log \beta(\alpha_j)}{\partial \alpha} \\
&= \frac{\partial}{\partial \alpha_j}\left[\log \frac{\prod_{i=1}^{k}\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{k}\alpha_i)}\right] \\
&= \sum_{i=1}^{k} \frac{\partial}{\partial \alpha_j}\left[\log \frac{\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{k}\alpha_i)}\right] \\
&= \sum_{i=1}^{k} \frac{\partial}{\partial \alpha_j}\left[\log\Gamma(\alpha_i) - \log\Gamma(\sum_{i=1}^{k}\alpha_i)\right] \\
&= \psi(\alpha_j) - \psi(\sum_{i=1}^{k}\alpha_i)
\end{aligned}$$

D.3. Multivariate Normal

Standard Form

$$p(x|\mu,\Sigma) = |2\pi\Sigma|^{-\frac{1}{2}} \exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)) \tag{D.8}$$

Exponential Form

$$\begin{aligned}
p(x|\mu,\Sigma) = &\exp\{\langle \mathrm{vec}(-\frac{1}{2}\Sigma^{-1}), \mathrm{vec}(xx^T)\rangle + \langle \Sigma^{-1}\mu, x\rangle \\
&- \frac{1}{2}\left[d\log 2\pi + \log|\Sigma| + \mu^T\Sigma^{-1}\mu\right]\}
\end{aligned}$$

where vec standing for vectorizing, i.e. flatten matrix into single vector.

$$\begin{aligned}
&\eta_1 = \Sigma^{-1} && T_1 = x && \mu = -\frac{1}{2}\eta_2^{-1}\eta_1 \\
&\eta_2 = vec(-\frac{1}{2}\Sigma^{-1}) && T_2 = vec(xx^T) && \Sigma = -\frac{1}{2}\eta_2^{-1}
\end{aligned}$$

The following gives the exponential form:

$$\begin{aligned}
\exp\log p(x|\mu,\Sigma) &= \exp\{\log|2\pi\Sigma|^{-\frac{1}{2}}\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))\} \\
&= \exp-\frac{1}{2}\Big((x-\mu)^T\Sigma^{-1}(x-\mu)+\log 2\pi^d+\log|\Sigma|\Big) \\
&= \exp\{-\frac{1}{2}\Big((x^T\Sigma^{-1}x-\mu^T\Sigma^{-1}x-x^T\Sigma^{-1}\mu+\mu^T\Sigma^{-1}\mu) \\
&\quad +\log 2\pi^d+\log|\Sigma|\Big)\} \\
&= \exp\{\langle-\frac{1}{2}\Sigma^{-1},xx^T\rangle_F+\langle\Sigma^{-1}\mu,x\rangle \\
&\quad -\frac{1}{2}\Big[d\log 2\pi+\log|\Sigma|+\mu^T\Sigma^{-1}\mu\Big]\} \\
&= \exp\{\langle\mathrm{vec}(-\frac{1}{2}\Sigma^{-1}),\mathrm{vec}(xx^T)\rangle+\langle\Sigma^{-1}\mu,x\rangle \\
&\quad -\frac{1}{2}\Big[d\log 2\pi+\log|\Sigma|+\mu^T\Sigma^{-1}\mu\Big]\}
\end{aligned}$$

which then gives:

$$A(\eta) = -\frac{1}{4}\eta_1^T\eta_2^{-1}\eta_1-\frac{1}{2}\log|-2\eta_2|$$

Expected Statistics

$$\mathbb{E}[x] = \mu \tag{D.9}$$

$$\mathbb{E}[xx^T] = \frac{\delta A(\eta)}{\delta\eta_2} \tag{D.10}$$

As according to the cumulant generating function properties we can find

$$\begin{aligned}
\mathbb{E}[x] &= \frac{\delta A(\eta)}{\delta\eta_1} \\
&= \frac{\delta}{\delta\eta_1}-\frac{1}{4}\eta_1^T\eta_2^{-1}\eta_1-\frac{1}{2}\log|-2\eta_2| \\
&= \frac{\delta}{\delta\eta_1}-\frac{1}{4}\eta_1^T\eta_2^{-1}\eta_1 \\
&= -\frac{1}{2}\eta_1^T\eta_2^{-1} \\
&= \frac{1}{2}(\Sigma^{-1}\mu)^T*2\Sigma \\
&= \mu
\end{aligned}$$

D.4. Matrix Normal

Standard Form

$$p(X|M,U,V) = \frac{\exp\left(-\frac{1}{2}\mathrm{Tr}\left[V^{-1}(X-M)^T U^{-1}(X-M)\right]\right)}{(2\pi)^{np/2}|V|^{n/2}|U|^{p/2}} \tag{D.11}$$

The Matrix Normal and Multivariate Normal are equivalent in the following way:

$$X \sim \mathcal{MN}(M,U,V) \tag{D.12}$$

if and only if

$$\mathrm{vec}(X) \sim \mathcal{N}(\mathrm{vec}(M), V \otimes U) \tag{D.13}$$

First, focus on the numerator:

$$\begin{aligned}
&-\frac{1}{2}\mathrm{Tr}\left[V^{-1}(X-M)^T U^{-1}(X-M)\right] \\
&= -\frac{1}{2}\mathrm{Tr}\left[(X-M)^T U^{-1}(X-M)V^{-1}\right] \\
&= -\frac{1}{2}\langle (X-M), U^{-1}(X-M)V^{-1}\rangle_F \\
&= -\frac{1}{2}\mathrm{vec}(X-M)^T \mathrm{vec}(U^{-1}(X-M)V^{-1}) \\
&= -\frac{1}{2}\mathrm{vec}(X-M)^T \left(V^{-1}\otimes U^{-1}\right)\mathrm{vec}(X-M) \\
&= -\frac{1}{2}\mathrm{vec}(X-M)^T (V\otimes U)^{-1}\,\mathrm{vec}(X-M) \\
&= -\frac{1}{2}(\mathrm{vec}(X)-\mathrm{vec}(M))^T (V\otimes U)^{-1}(\mathrm{vec}(X)-\mathrm{vec}(M))
\end{aligned}$$

which in order of steps is, cyclic property of the trace, trace equivalent to Frobenius inner product, Frobenius inner product is equivalent to the product of vectorized elements, vectorization of matrix multiplication, property of the Kronecker product, property of vectorization.

Second, for the denominator we can use the determinant property:

$$|V\otimes U| = |V|^n |U|^p$$

D.5. Inverse Wishart

Standard Form

$$p(x|\Psi,\nu) = \frac{|\Psi|^{\frac{\nu}{2}}}{2^{\frac{\nu p}{2}}\Gamma_p(\frac{\nu}{2})}|x|^{-(\nu+p+1)/2}\exp\{-\frac{1}{2}\mathrm{Tr}(\Psi x^{-1})\} \tag{D.14}$$

where Γ_p is the multivariate gamma function.

Exponential Form

$$p(x|\Psi,\nu) = \exp\{\langle -\frac{(\nu+p+1)}{2}, \log|x|\rangle + \langle -\frac{1}{2}\Psi, x^{-1}\rangle_F - \log\frac{2^{\frac{\nu p}{2}}\Gamma_p(\frac{\nu}{2})}{|\Psi|^{\frac{\nu}{2}}}\} \tag{D.15}$$

with

$$\eta_1 = -\frac{v+p+1}{2} \qquad T_1 = log|x|$$
$$\eta_2 = -\frac{1}{2}\Psi \qquad T_2 = x^{-1}$$

The exponential form is derived as follows:

$$\begin{aligned}
\exp\{\log p(x|\Psi,\nu)\} &= \exp\{\log\frac{|\Psi|^{\frac{\nu}{2}}}{2^{\frac{\nu p}{2}}\Gamma_p(\frac{\nu}{2})}|x|^{-(\nu+p+1)/2}\exp\{-\frac{1}{2}\mathrm{Tr}(\Psi x^{-1})\}\} \\
&= \exp\{\log|x|^{-(\nu+p+1)/2}\exp\{-\frac{1}{2}\mathrm{Tr}(\Psi x^{-1})\} \\
&\quad -\log\frac{2^{\frac{\nu p}{2}}\Gamma_p(\frac{\nu}{2})}{|\Psi|^{\frac{\nu}{2}}}\} \\
&= \exp\{-\frac{(\nu+p+1)}{2}\log|x| - \frac{1}{2}\mathrm{Tr}(\Psi x^{-1}) \\
&\quad -\log\frac{2^{\frac{\nu p}{2}}\Gamma_p(\frac{\nu}{2})}{|\Psi|^{\frac{\nu}{2}}}\} \\
&= \exp\{\langle -\frac{(\nu+p+1)}{2}, \log|x|\rangle + \langle -\frac{1}{2}\Psi, x^{-1}\rangle_F \\
&\quad -\log\frac{2^{\frac{\nu p}{2}}\Gamma_p(\frac{\nu}{2})}{|\Psi|^{\frac{\nu}{2}}}\}
\end{aligned} \tag{D.16}$$

D.6. Normal Inverse Wishart

We have the following generative process

$$\begin{aligned} \Sigma &\sim W^{-1}(\nu, \Psi) \\ \mu|\Sigma &\sim N(\mu_0, \kappa^{-1}\Sigma) \end{aligned} \tag{D.17}$$

for which the pdfs can be found in section D.5 and D.4 respectively.

Standard Form

$$\begin{aligned} p(\mu, \Sigma|\mu_0, \kappa, \nu, \Psi) =& N(\mu_0, \kappa^{-1}\Sigma)W^{-1}(\nu, \Psi) \\ =& \frac{|\Psi|^{\frac{\nu}{2}}|\Sigma|^{-(\nu+p+1)/2}}{2^{\frac{\nu p}{2}}\Gamma_p(\frac{\nu}{2})|2\pi\kappa^{-1}\Sigma|^{\frac{1}{2}}} \exp\{-\frac{1}{2}\mathrm{Tr}(\Psi\Sigma^{-1}) \\ & -\frac{1}{2}(\mu-\mu_0)^T\kappa\Sigma^{-1}(\mu-\mu_0)\} \end{aligned} \tag{D.18}$$

Exponential Form

$$\begin{aligned} p(\mu, \Sigma|\mu_0, \kappa, \nu, \Psi) \propto \exp\{&\langle -\frac{1}{2}\log|\Sigma|, \nu+p+2\rangle \\ &+\langle -\frac{1}{2}\Sigma^{-1}, \Psi+\kappa\mu_0\mu_0^T\rangle \\ &+\langle \Sigma^{-1}\mu, \kappa\mu_0\rangle \\ &+\langle -\frac{1}{2}\mu^T\Sigma^{-1}\mu, \kappa\rangle\} \end{aligned} \tag{D.19}$$

with normalization constant:

$$\log Z = \log \frac{|\Psi|^{\frac{\nu}{2}}}{2^{\frac{\nu p}{2}}\Gamma_p(\frac{\nu}{2})(2\pi\kappa^{-1})^{\frac{p}{2}}} \tag{D.20}$$

$$\begin{aligned} \eta_1 &= \nu+p+2 \\ \eta_2 &= \Psi+\kappa\mu_0\mu_0^T \\ \eta_3 &= \kappa\mu_0^T \\ \eta_4 &= \kappa \end{aligned} \qquad \begin{aligned} T_1 &= -\frac{1}{2}\log|\Sigma| \\ T_2 &= -\frac{1}{2}\Sigma^{-1} \\ T_3 &= \Sigma^{-1}\mu \\ T_4 &= -\frac{1}{2}\mu^T\Sigma^{-1}\mu \end{aligned} \qquad \begin{aligned} \kappa &= \eta_4 \\ \mu_0 &= \frac{\eta_3}{\eta_4} \\ \Psi &= \eta_2 - \frac{\eta_3\eta_3^T}{\eta_4} \\ \nu &= \eta_1 - p - 2 \end{aligned}$$

$$
\begin{aligned}
p(\mu, \Sigma | \mu_0, \kappa, \nu, \Psi) &\propto |\Sigma|^{-(\nu+p+2)/2} \exp\{-\frac{1}{2}\operatorname{Tr}(\Psi\Sigma^{-1}) - \frac{\kappa}{2}(\mu-\mu_0)^T\Sigma^{-1}(\mu-\mu_0)\} \\
&\propto \exp\{\log|\Sigma|^{-(\nu+p+2)/2} - \frac{1}{2}\operatorname{Tr}(\Psi\Sigma^{-1}) - \frac{\kappa}{2}(\mu-\mu_0)^T\Sigma^{-1}(\mu-\mu_0)\} \\
&\propto \exp\{\langle -\frac{1}{2}\log|\Sigma|, \nu+p+2\rangle \\
&\qquad + \langle -\frac{1}{2}\Sigma^{-1}, \Psi\rangle_F \\
&\qquad + \langle -\frac{1}{2}\Sigma^{-1}, \kappa\mu_0\mu_0^T\rangle \\
&\qquad + \langle \Sigma^{-1}\mu, \kappa\mu_0\rangle \\
&\qquad + \langle -\frac{1}{2}\mu^T\Sigma^{-1}\mu, \kappa\rangle\} \\
&\propto \exp\{\langle -\frac{1}{2}\log|\Sigma|, \nu+p+2\rangle \\
&\qquad + \langle -\frac{1}{2}\Sigma^{-1}, \Psi+\kappa\mu_0\mu_0^T\rangle \\
&\qquad + \langle \Sigma^{-1}\mu, \kappa\mu_0\rangle \\
&\qquad + \langle -\frac{1}{2}\mu^T\Sigma^{-1}\mu, \kappa\rangle\}
\end{aligned}
$$

Again we can rewrite the normalization constant as a function of the natural parameters instead

$$
A(\eta) = \log \frac{|\eta_2 - \frac{\eta_3\eta_3^T}{\eta_4}|^{\frac{\eta_1-p-2}{2}}}{2^{\frac{(\eta_1-p-2)p}{2}}\Gamma_p(\frac{\eta_1-p-2}{2})(2\pi\eta_4^{-1})^{\frac{p}{2}}} \tag{D.21}
$$

Expected Statistics

$$
\mathbb{E}[T_2] = \frac{\nu}{2}\Psi^{-1} \tag{D.22}
$$

$$
\mathbb{E}[T_3] = -2\,\mathbb{E}[T_2]\mu_0 \tag{D.23}
$$

$$
\mathbb{E}[T_4] = \mu_0^T\frac{\mathbb{E}[T_3]}{-2} + \frac{p}{2\kappa} \tag{D.24}
$$

$$
\mathbb{E}[T_1] = \frac{1}{2}\log|\Psi| - \frac{p}{2}\log 2 - \frac{1}{2}\psi_p(\frac{\nu}{2}) \tag{D.25}
$$

$$
\begin{aligned}
\mathbb{E}[T_2] &= \frac{\delta A(\eta)}{\delta \eta_2} \\
&= \frac{\eta_1 - p - 2}{2} \frac{\delta}{\delta \eta_2} \left[\log |\eta_2 - \frac{\eta_3 \eta_3^T}{\eta_4}| \right] \\
&= \frac{\eta_1 - p - 2}{2} \left(\eta_2 - \frac{\eta_3 \eta_3^T}{\eta_4} \right)^{-1} \\
&= \frac{\nu}{2} \Psi^{-1}
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}[T_3] &= \frac{\delta A(\eta)}{\delta \eta_3} \\
&= \frac{\eta_1 - p - 2}{2} \frac{\delta}{\delta \eta_3} \left[\log |\eta_2 - \frac{\eta_3 \eta_3^T}{\eta_4}| \right] \\
&= \frac{\eta_1 - p - 2}{2} \left(\eta_2 - \frac{\eta_3 \eta_3^T}{\eta_4} \right)^{-1} \left(-2 \frac{\eta_3}{\eta_4} \right) \\
&= -2 \frac{\nu}{2} \Psi^{-1} \mu_0 \\
&= -2 \, \mathbb{E}[T_2] \mu_0
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}[T_4] &= \frac{\delta A(\eta)}{\delta \eta_4} \\
&= \frac{\eta_1 - p - 2}{2} \frac{\delta}{\delta \eta_4} \left[\log |\eta_2 - \frac{\eta_3 \eta_3^T}{\eta_4}| \right] - \frac{p}{2} \frac{\delta}{\delta \eta_4} \log(\frac{2\pi}{\eta_4}) \\
&= \frac{\eta_1 - p - 2}{2} \frac{\eta_3^T}{\eta_4} \left(\eta_2 - \frac{\eta_3 \eta_3^T}{\eta_4} \right)^{-1} \frac{\eta_3}{\eta_4} + \frac{p}{2\eta_4} \\
&= \frac{\eta_3^T}{\eta_4} \frac{\mathbb{E}[T_3]}{-2} + \frac{p}{2\eta_4} \\
&= \mu_0^T \frac{\mathbb{E}[T_3]}{-2} + \frac{p}{2\kappa}
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}[T_1] &= \frac{\delta A(\eta)}{\delta \eta_1} \\
&= \frac{\delta}{\delta \eta_1} \frac{\eta_1 - p - 2}{2} \log |\Psi| - \frac{(\eta_1 - p - 2)p}{2} \log 2 - \log \Gamma_p(\frac{\eta_1 - p - 2}{2}) \\
&= \frac{1}{2} \log |\Psi| - \frac{p}{2} \log 2 - \frac{1}{2} \psi_p(\frac{\nu}{2})
\end{aligned}
$$

Bibliography

[Ama98] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[DA15] David Duvenaud and Ryan P Adams. Black-box stochastic variational inference in five lines of python. In *NIPS Workshop on Black-box Learning and Inference*, 2015.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[HBWP13] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 2013.

[JDW+16] Matthew J Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. *Advances in neural information processing systems*, 29, 2016.

[JGJS99] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[MK07] Geoffrey J McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

[PBJ12] John Paisley, David Blei, and Michael Jordan. Variational bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*, 2012.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca

Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[WBJ05] John Winn, Christopher M Bishop, and Tommi Jaakkola. Variational message passing. *Journal of Machine Learning Research*, 6(4), 2005.

[WJ+08] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.