Interpretable Models in Probabilistic Machine Learning

Hyun Jik Kim

Worcester College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2019

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Yee Whye Teh, for his support, advice and inspiration over the course of my Ph.D. It was a great privilege to be able to learn from such an academic, whose sheer breadth of knowledge is astounding and the insight that holds together these pieces of knowledge I find even more remarkable. Personally, his advice for research and meta-research has been of tremendous value in my growth as an independent researcher, and his passion for the discipline will remain with me for a long time. I was also inspired by the care and time he offers to his students, for which I feel both indebted and intimidated, as he has set the bar very high for being an exemplary supervisor. If ever I have the chance to supervise junior researchers or students in the future, I will constantly find myself asking 'what would Yee Whye do?'

Moreover, I would like to whole-heartedly thank all my collaborators Andriy Mnih, Adam Kosiorek, Xiaoyu Lu, Hongseok Yang, Jonathan Schwarz, Marta Garnelo, Tuan Anh Le, Ali Eslami, Seth Flaxman, Dan Rosenbaum and Oriol Vinyals, from whom I have received a lot of help and learned a great deal. I would especially like to thank Andriy, whose constructive criticism and meticulous attention to detail helped significantly raise the quality of my latest works, and also for being a most reliable go-to person for all kinds of advice throughout my time at DeepMind.

I am also grateful to my friends and colleagues from Oxford, Cambridge, London and Korea whose presence and words generated great happiness, laughter and support during my time at Oxford. I am especially indebted to the members of the Oxford Korean Society Football Club, its Cambridge counterpart, and Worcester College Association Football Club for taking care of my physical and mental health on and off the pitch, respectively.

Finally, I would like to thank Soohyun for enriching my time at Oxford, and above all, my beloved family who have always shown unquestioning support.

Abstract

This thesis describes contributions to the field of interpretable models in probabilistic machine learning, by first outlining the desiderata and properties associated with the term *interpretability*. We claim that probabilistic models are suitable candidates for interpretable machine learning, and this claim is supported by examples of such models that satisfy two key properties of interpretability: *transparency*, that offers an understanding of the model's mechanism, and *post-hoc interpretability*, that gives other useful information about the model after training, such as explaining its predictions. Henceforth, we introduce relevant background literature in probabilistic machine learning, focusing on Bayesian inference of probabilistic models. Armed with these pre-requisites, we proceed to describe examples of probabilistic models that enjoy various interpretable properties. First, we propose a method for regression that is motivated from Gaussian Processes (GPs), that has applications to collaborative filtering with side-information and generalises classic probabilistic matrix factorisation methods in this context. Second, we develop a scalable algorithm for automated GP model selection, whereby the form of the selected GP models allows them to be translated into a natural language description of its properties. Third, we introduce a Variational Autoencoder (VAE) model that can disentangle independent factors of variations in a dataset of images by learning a factorisable latent distribution in an unsupervised fashion. Finally, we describe a model that can learn stochastic processes in a data-driven fashion with deep architectures by using the concept of *attention*.

Publications

Most chapters of this integrated thesis are publications at Machine Learning conferences. At the end of each chapter we include a description of the author's contribution. For convenience we detail the relevant publications here.

Chapter 3 contains

Kim, H., Lu, X., Flaxman, S. and Teh, Y.W., 2016, May. Collaborative Filtering with Side Information: a Gaussian Process Perspective. arXiv preprint arXiv:1605.07025.

Chapter 4 contains

Kim, H. and Teh, Y.W., 2018, April. Scaling up the Automatic Statistician: Scalable Structure Discovery using Gaussian Processes. In Artificial Intelligence and Statistics (AISTATS).

Chapter 5 contains

Kim, H. and Mnih, A., 2018, July. Disentangling by Factorising. In Proceedings of the 35th International Conference on Machine Learning (ICML).

Chapter 6 contains

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, M., Vinyals, O. and Teh, Y.W., 2019, May. Attentive Neural Processes. In Proceedings of the International Conference on Learning Representations (ICLR).

Contents

Chapter 1

Introduction

Recent years have seen machine learning (ML) systems being deployed to solve real-world problems, such as voice recognition [Hinton et al., 2012], language translation Bahdanau et al. [2014], item (video / product) recommendation to users [Koren et al., 2009] and autonomous driving [Menze and Geiger, 2015] to name a few. While many systems have been successful and proved to outperform state of the art, many also have failure modes that are constantly exposed by their users. These range from minor mistakes in translation or object recognition to more serious issues that give rise to ethical problems or pose threats to human lives. These issues have called for *interpretable* models and algorithms, whose mechanisms and decisions we can better understand [Doshi-Velez, 2017]. If a thorough understanding is difficult, especially with black-box algorithms whose predictions show high accuracy but are difficult to explain, then models that can provide users with an idea of uncertainty in the prediction can be a reasonable alternative (e.g. Kendall and Gal [2017]). Hence probabilistic ML, or Bayesian ML, which offer coherent and principled frameworks to reason about uncertainty, are valuable areas of research that provide users with such models.

This thesis describes examples of such models in probabilistic ML, with a particular focus on their interpretable properties. This is an integrated thesis, where Chapters 3-6 are in the form of self-contained papers. Chapter 2 introduces the necessary pre-requisites in Bayesian machine learning for the understanding of the following chapters, focusing on exact and approximate posterior inference using MCMC and Variational Inference (VI). We also provide examples of models such as Gaussian Processes (GPs) and Probabilistic Matrix Factorisation (PMF) models along with different methods of inference and optimisation for these models. We proceed to introduce interpretability

in ML and how each subsequent chapter fits into this landscape.

Chapter 3 introduces a new model for Collaborative Filtering (CF) with side information (about users/items) that is motivated from Gaussian Process (GP) regression with low-rank matrix factorisation. Our model generalises classical Bayesian matrix factorisation models for CF, and can also be applied to general regression problems. It is particularly suitable for grid-structured data, where the dependence across different covariates are close to being separable. The model is interpretable in the sense that the regression function is additive, with the additive components arising from a product of features of each covariate. Thus the model allows us to analyse each additive component and investigate the multiplicative interactions between the covariates within the component.

Chapter 4 introduces an algorithm for model selection that builds on the Automatic Statistician (AS) [1], an algorithm for automated exploratory data analysis. Given some low-dimensional regression data, AS first fits a GP to the data with an algorithm for automatically choosing a suitable parametric form of the kernel. This is called Compositional Kernel Search (CKS) [Duvenaud et al., 2013], whose distinctive feature (compared to other GP model selection approaches) is that the resulting kernels are interpretable, i.e. the GP model can be translated into a natural language report. One drawback of CKS is that it does not scale to larger datasets due to its $O(N^3)$ computational complexity for GP inference. Our work proposes a modified algorithm, called Scalable Kernel Composition (SKC), reduces the algorithmic complexity to $O(N^2)$ while also preserving interpretability. In doing so, we derive a novel cheap upper bound to the GP marginal likelihood that sandwiches the true value with the known variational lower bound. We show that the upper bound is considerably tighter than the lower bound, and plays a crucial role for model selection.

Chapter 5 proposes FactorVAE, a generative model-based method for unsupervised learning of disentangled representations on data generated from independent factors of variation. The motivation for the method covers all modalities of data, but it was evaluated mainly on image data owing to ease of displaying latent traversals. The method uses the Variational Autoencoder (VAE) [Kingma and Welling, 2013] framework with an encoder that maps an image to a lower-dimensional latent vector, which corresponds to the representation of the image. The decoder (that forms the generative model) maps the latents back to the image space, and the model is learned by maximising a lower bound on the log marginal likelihood of the training

[1] `http://www.automaticstatistician.com/index/`

images. The aim is to learn an encoder such that the different factors of variation of each image are distilled into each dimension of the latent vector, hence the term *disentangling*. We propose to achieve this by encouraging the distribution of the latent vector to be factorial via a *Total Correlation* (TC) penalty appended to the VAE loss, and also show that this can allow learning more useful generative models than previous approaches to unsupervised disentangling. We also introduce a new metric for quantifying disentangling that addresses the problems of a commonly used metric in Higgins et al. [2016].

Finally Chapter 6 introduces Attentive Neural Processes (ANPs), a deep learning model that is used to learn stochastic processes in a data-driven fashion. It extends Neural Processes (NPs) (Garnelo et al. [2018b]) by incorporating an attention mechanism that remedies the issue of underfitting observed in NPs. The attention mechanism allows each target input location to attend to the observed data that is relevant for the output prediction. We show that this greatly improves the accuracy of predictions, results in noticeably faster training, and expands the range of functions that can be modelled.

Chapter 2

Probabilistic Machine Learning: Background

In this chapter we present a brief introduction to probabilistic machine learning, focusing on the background that is relevant for understanding the body of the thesis. We give a quick overview of machine learning basics with linear regression and kernels, then delve into more detail to explain Bayesian machine learning. We cover priors, likelihoods and posterior inference, in particular MCMC and variational inference, along with Bayesian model selection. We flesh out these concepts and methodologies with concrete examples of probabilistic models such as Gaussian Processes and probabilistic matrix factorisation. We point out how each section relates to the subsequent chapters of the thesis.

2.1 Machine Learning Basics

Machine learning can be defined as a set of methods that can detect patterns in data, and use them to predict future data or perform decision making [Murphy, 2012a]. The data can either come in the form of input-output pairs, usually denoted by $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, or just the inputs $\{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$. This data is used to train a machine learning system for the aforementioned purposes, hence we refer to it as the *training set*. In the scenario with input-output data we perform *supervised learning*, where the goal is to learn a mapping from $\boldsymbol{x}$ to y that can generalise beyond the examples seen in the training set. The task is called *regression* in the case where y is real-valued, and *classification* in the case where y takes on discrete values. For the scenario with just input data we perform *unsupervised learning*, where the goal is to find interesting

patterns in the data that can help understand the data and/or make it useful for some downstream task. In either case, a standard probabilistic approach is to learn a distribution of the data, whether it be conditional $p(y|\boldsymbol{x})$ or unconditional $p(\boldsymbol{x})$. This is referred to as the probabilistic *model* that we are interested in learning. Usually the distribution is parameterised with some variables $\boldsymbol{\theta}$ as $p(y|\boldsymbol{x}, \boldsymbol{\theta})$, and the goal is to learn $\boldsymbol{\theta}$ from the data. The parameters $\boldsymbol{\theta}$ can either be deterministic or random, in which case you would learn a distribution over $\boldsymbol{\theta}$. A simple example is *Linear Regression*, which we outline below.

2.1.1 Linear Regression

In linear regression, we assume that the output y can be well approximated by a linear transformation of the output $\boldsymbol{x}$, with some random noise accounting for the error. Hence y is modelled as following a Gaussian distribution with fixed variance and the mean being a linear function of the input $\boldsymbol{x}$. This can be written as $p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\boldsymbol{w}^\top \boldsymbol{x}, \sigma^2)$, with parameters $\boldsymbol{\theta} = (\boldsymbol{w}, \sigma)$. A common method to learn the parameters of a model is by *maximum likelihood estimation* (MLE). In the case of linear regression, it is also known as *least squares*, for reasons which will become apparent later on. MLE learns the parameter by setting it to be the value that maximises the likelihood of the data. This value also maximises the log likelihood, due to the monotonicity of the logarithm. Assuming that the $y^{(i)}$ given $\boldsymbol{x}^{(i)}$ are independent, the log likelihood of the data can be written as follows

$$\begin{aligned} l(\theta) &\coloneqq \sum_{i=1}^{N} \log p(y^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) \\ &= \sum_{i=1}^{N} \log \Big[\big(2\pi\sigma^2\big)^{-\frac{1}{2}} \exp\Big(-\frac{1}{2\sigma^2}(y^{(i)} - \boldsymbol{w}^\top \boldsymbol{x}^{(i)})^2 \Big) \Big] \\ &= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y^{(i)} - \boldsymbol{w}^\top \boldsymbol{x}^{(i)})^2. \end{aligned} \tag{2.1}$$

Minimising this quantity with respect to the weights $\boldsymbol{w}$ amounts to minimising the least squares term on the right of Equation (2.1), which makes intuitive sense. With basic calculus, it is easy to derive the maximum likelihood estimates of $\boldsymbol{w}$ and σ in closed form: $\hat{\boldsymbol{w}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}$, $\hat{\sigma}^2 = \frac{1}{N}||\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{w}}||^2$ where $\mathbf{X}$ is a matrix whose ith row is $\boldsymbol{x}^{(i)}$ and $\boldsymbol{y}$ is a vector with entries $y^{(i)}$. The fitted values $\hat{\boldsymbol{y}} \coloneqq \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}$ can be interpreted as a linear projection of $\boldsymbol{y}$ onto the column space of $\mathbf{X}$ (the vector space spanned by the columns of $\mathbf{X}$), which again makes sense in retrospect since $\mathbf{X}\boldsymbol{w}$

lives in this column space $\forall \boldsymbol{w}$. In fact, the Gaussian assumption on y is not strictly necessary for obtaining the estimator $\hat{\boldsymbol{w}}$. The *Gauss-Markov Theorem* [Gauss, 1823] tells us that for the model $\boldsymbol{y} = \boldsymbol{w}^\top \mathbf{X} + \boldsymbol{\epsilon}$ with $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and $\mathrm{Var}[\boldsymbol{\epsilon}] = I$, the above value of $\hat{\boldsymbol{w}}$ is the *best linear unbiased estimator* (BLUE), in the sense that for any unbiased estimator $\tilde{\boldsymbol{w}}$, $\mathrm{Var}[\tilde{\boldsymbol{w}}] - \mathrm{Var}[\boldsymbol{w}]$ is positive semi-definite.

Sometimes we would want to model non-linear relations between $\boldsymbol{x}$ and y, in which case we can apply linear regression to non-linear features $\boldsymbol{\phi}(\boldsymbol{x})$ instead of $\boldsymbol{x}$. In other words, we regress $\boldsymbol{y}$ on $\boldsymbol{\Phi}$ (matrix whose ith row is $\boldsymbol{\phi}(\boldsymbol{x}^{(i)})$) instead of $\mathbf{X}$, and the same formulae for MLE hold with $\mathbf{X}$ replaced by $\boldsymbol{\Phi}$. One method to construct such features is by using *kernels*.

2.1.2 Kernels

A *kernel* function $k(\boldsymbol{x}, \boldsymbol{x}')$ is a real, scalar valued function of two arguments $\boldsymbol{x}, \boldsymbol{x}'$ living in an arbitrary space $\mathcal{X}$. Typically it is symmetric, so can be interpreted as a measure of similarity between two points in $\mathcal{X}$, as is done in Chapter 3, but this is not strictly necessary.

The most common example of a kernel is the *squared exponential (SE) kernel*, also known as the *Gaussian kernel*

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^\top \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{x}') \right). \tag{2.2}$$

If Σ is diagonal, then the expression inside the exponent can be expressed as a sum over the different dimensions

$$k(x, x') = \exp\left(-\sum_j \frac{1}{2\sigma_j^2}(x_j - x_j')^2 \right). \tag{2.3}$$

The σ_j is also referred to as the length scale of the jth dimension of the x domain, since it represents the extent to which the kernel value changes as x_j changes. So dimensions with relatively large σ_j have a small effect on the kernel, thus such dimensions can usually be ignored when the kernel has been used to model the data. Hence this kernel is also known as the *automatic relevance determination (ARD) kernel*. In the case where the length scales are equal across all dimensions, we have the *isotropic kernel*. The above kernels are *stationary*, as the kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ is a function of $(\boldsymbol{x} - \boldsymbol{x}')$, hence invariant to translations in the input space.

In some contexts the kernel k satisfies the condition that the *Gram matrix* $\mathbf{K}$ is positive definite where $K_{ij} \coloneqq k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ for all possible inputs $(\boldsymbol{x}^{(i)})_{i=1}^N$. Such a

kernel is called a *positive definite kernel* or *Mercer kernel*. The interesting property of such kernels is that the Gram matrix allows an eigen-decomposition $\mathbf{K} = \mathbf{U}^\top \Lambda \mathbf{U}$ where Λ is a diagonal matrix of eigenvalues. Then we can write $\mathbf{K}_{ij} = \boldsymbol{\phi}(\boldsymbol{x}^{(i)})^\top \boldsymbol{\phi}(\boldsymbol{x}^{(j)})$ $\forall i, j$. In fact, Mercer's Theorem [König, 2013] tells for a positive definite kernel k that satisfies some mild conditions, there exists a (possibly infinite) sequence of functions $\boldsymbol{\phi}_i \colon \mathcal{X} \longrightarrow \mathbb{R}$ and positive real values λ_i such that $\sum_{i=1}^{N} \lambda_i \boldsymbol{\phi}_i(\boldsymbol{x}) \boldsymbol{\phi}_i(\boldsymbol{x}') \to k(\boldsymbol{x}, \boldsymbol{x}')$ as $n \to \infty$. The conditions and type of convergence are detailed in König [2013]. In fact, these $\boldsymbol{\phi}_i$ and λ_i are the eigenfunctions and eigenvalues of k i.e. $\int k(x, x') \boldsymbol{\phi}(\boldsymbol{x}) d\mu(\boldsymbol{x}) = \lambda \boldsymbol{\phi}(\boldsymbol{x}')$ where $(\mathcal{X}, \mu)$ is a finite measure space on which k is defined. Taking a finite approximation of the series, we can find finite dimensional feature maps $\boldsymbol{\phi}$ such that $k(\boldsymbol{x}, \boldsymbol{x}') \approx \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$ $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$. For some subsets of Mercer kernels, it is easy to find such feature maps as we will show later, but finding such feature maps for all Mercer kernels is difficult in general.

Defining a positive definite kernel from a feature map, however, is easy: $k(\boldsymbol{x}, \boldsymbol{x}') \coloneqq \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$. Also, we can easily construct new Mercer kernels from old via summation or multiplication: $k(\boldsymbol{x}, \boldsymbol{x}') \coloneqq k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}')$, $k(\boldsymbol{x}, \boldsymbol{x}') \coloneqq k_1(\boldsymbol{x}, \boldsymbol{x}') \times k_2(\boldsymbol{x}, \boldsymbol{x}')$. The resulting kernels can also be represented as an inner product of feature maps, by concatenating the feature maps of the original kernels or taking their Kronecker product. Hence the new kernels are also positive definite. This observation will be used to search over new positive definite kernels from old in Chapter 4. Such features derived from kernels can be used to form non-linear features in standard machine learning models (such as linear regression in Section 2.1.1). For kernels whose feature maps are difficult to derive, a more direct method is to replace inner products in standard models with kernels. This is known as the *kernel trick*. This can be readily applied to models whose formulation or solution can be written as a function of inner products $\boldsymbol{x}^\top \boldsymbol{x}'$, by replacing it with $k(\boldsymbol{x}, \boldsymbol{x}')$. This trick leads to various *kernelised* algorithms such as *kernelised k-NN* [Yu et al., 2002], *kernelised ridge regression* [Saunders et al., 1998] and *kernel PCA* [Schölkopf et al., 1997].

2.2 Bayesian Machine Learning

In Bayesian machine learning, we define a statistical model $p(\boldsymbol{x}, \boldsymbol{\theta})$ of the data $\boldsymbol{x}$, where $\boldsymbol{\theta}$ are random latent variables that must be inferred from the data. One of the reasons that we assume a distribution over $\boldsymbol{\theta}$ is to model uncertainty in the parameters, that will be reflected in the uncertainties for making predictions. For

sensible decision making to be possible using such models, accurate quantification of predictive uncertainty is crucial. $\boldsymbol{\theta}$ can either be 1) a set of *global latents*, in which case $\boldsymbol{x}$ represents the dataset whose data points share the same latents, or 2) *local latents*, in which case $\boldsymbol{x}$ represents a single data point and different $\boldsymbol{x}$ depend on different $\boldsymbol{\theta}$, with a further joint distribution that defines their dependence structure. In either case, the model is composed of a likelihood $p(\boldsymbol{x}|\boldsymbol{\theta})$ and a prior distribution $p(\boldsymbol{\theta})$. The likelihood defines the distribution of the data given a fixed value of $\boldsymbol{\theta}$, whereas the prior defines a distribution over $\boldsymbol{\theta}$ alone. Once the model has been defined, we Bayesians are interested in the posterior distribution, given by *Bayes' Theorem*

$$p(\boldsymbol{\theta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}}. \tag{2.4}$$

The posterior is useful for obtaining the posterior predictive distribution $p(\boldsymbol{x}^*|\boldsymbol{x})$ of a new data point $\boldsymbol{x}^*$.

2.2.1 Prior

The prior over latent variables in a Bayesian model provides an inductive bias for the model, to prefer some values of $\boldsymbol{\theta}$ over others, regardless of the data. Hence the prior allows us to incorporate prior beliefs about the latents previous to seeing the data. This helps learn faster from smaller amounts of data. In this subsection we survey the different types of priors that are commonly used.

Conjugate Prior

The most commonly used prior in Bayesian machine learning is the *conjugate prior*. This is a choice of prior that allows the prior and posterior to be in the same family of distributions, given the likelihood. This circumvents the common difficulty of Bayesian inference, which is the difficulty of computing the posterior distribution due to the intractable normalising constant $\int p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$, that is often required to infer the predictive distribution (Equation 2.6). The mean of this distribution is usually set to be the prediction, and its variance is used for reporting error bars. For example, given a Bernoulli likelihood $X^{(i)}|\boldsymbol{\theta} \overset{\text{iid}}{\sim} \text{Ber}(\theta)$, we have $p(\boldsymbol{x}|\theta) = \prod_{i=1}^{N} p(x^{(i)}|\theta) = \theta^{N_1}(1-\theta)^{N_2}$ where $N_1 = \sum_{i=1}^{N} \mathbb{I}(X_i = 1)$, $N_0 = \sum_{i=1}^{N} \mathbb{I}(X_i = 0)$, the counts of 1s and 0s respectively. In this case the unique conjugate prior is the $\text{Beta}(a, b)$ prior $p(\theta) \propto \theta^{a-1}(1-\theta)^{b-1}$, for which the posterior is $p(\theta|\boldsymbol{x}) \propto \theta^{a-1+N_1}(1-\theta)^{b-1+N_0}$. The parameters a, b of the prior are often referred to as *hyperparameters*. A concise interpretation is that the

posterior amounts to the prior with its hyperparameters updated by the counts of 1s and 0s.

Uninformative Prior

The prior happens to be one of the most criticised aspects of Bayesian machine learning, since the choice of prior gives subjectivity to the model, and one can get different predictions according to this choice. Choosing a conjugate prior for the sake of easy posterior computation is not free from this criticism. Hence an alternative method of choosing the prior is to minimise the effect of the prior, using *uninformative priors*.

The intuition for the uninformative prior is to 'let the data speak for itself'. The *uniform prior* where all values in the domain of θ are given equal weight is a naïve prior; for the Bernoulli-Beta model above, the Beta(1,1) prior, the uniform distribution over [0,1], leads to the posterior Beta$(N_1 + 1, N_0 + 1)$. This gives a posterior mean of $\frac{N_1+1}{N_1+N_0+2}$, whereas the MLE (c.f. Section 2.1.1) is $\frac{N_1}{N_1+N_0}$. Hence one could argue that the uniform prior is actually informative.

A different definition of an uninformative prior is one that leaves the prior invariant to reparameterisation, known as the *Jeffreys prior* [Jeffreys, 1961]. This can be derived by matching $p(\theta)$ to $p(\phi)$ where $\theta = h(\phi)$ is some reparameterisation of ϕ. Let us define $p(\phi) \propto (I(\phi))^{1/2}$ where $(I(\phi))$ is the *Fisher information*

$$I(\phi) \coloneqq -\mathbb{E}_{p(X)}\Big[\Big(\frac{d \log p(X|\phi)}{d\phi}\Big)^2\Big]. \tag{2.5}$$

It is known as a measure of information that the random variable X carries about the latent ϕ of the model at a given value of ϕ. Intuitively, if the derivative of the likelihood with respect to ϕ is high in magnitude, then the likelihood of X is sensitive to changes in ϕ, hence X carries substantial information about ϕ.

With such a prior on ϕ, it is easy to show that for any reparameterisation $\theta = h(\phi)$, we have that $\mathcal{I}(\theta)^{1/2} = \mathcal{I}(\phi)^{1/2}|\frac{d\phi}{d\theta}|$. With the change of variables formula applied to the prior, we have $p(\theta) = p(\phi)|\frac{d\phi}{d\theta}| \propto \mathcal{I}(\theta)^{1/2}$. Hence the Jeffreys prior is uninformative in the sense that the prior is invariant to reparameterisation. It is easy to show that the Jeffreys prior for the Beta-Bernoulli model above is Beta$(1/2, 1/2)$.

Improper Prior

An orthogonal class of priors are *improper priors*, which do not integrate to 1. These are usually defined by setting $p(\theta) \propto f(\theta)$ where $\int f(\theta)d\theta$ is infinite, hence the term

improper, but assume that the factor of proportion cancels in the numerator and denominator of the posterior (c.f. Equation 2.4), to define a proper posterior. An example of an improper prior is the uniform prior over an infinite interval (e.g. $\mathbb{R}$), defined as $p(\theta) \propto 1$.

2.2.2 Exact Posterior Inference: Bayesian Linear Regression

The posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x})$ can be interpreted as a summary of our knowledge of the latent variable given the data and the probabilistic model $p(\boldsymbol{x}, \boldsymbol{\theta})$. The posterior distribution is particularly useful for computing or approximating the *posterior predictive distribution*

$$p(\boldsymbol{x}^*|\boldsymbol{x}) = \int p(\boldsymbol{x}^*|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{x}) d\boldsymbol{\theta}. \tag{2.6}$$

This allows us to quantify both the mean prediction as well as the predictive variance, the uncertainty in the prediction.

The problem of computing the posterior, also referred to as *posterior inference*, can in some cases be simple. For example with a conjugate prior, the posterior distribution can be computed analytically, as shown above for the Beta-Bernoulli model. Another standard example of such conjugate priors is Bayesian linear regression with a Gaussian prior on the linear coefficients $\boldsymbol{w}$, relevant for Chapter 3. If we assume $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|0, I)$ and Gaussian likelihood $p(y|\boldsymbol{x}, \boldsymbol{w}, \sigma^2) = \mathcal{N}(y|\boldsymbol{w}^\top \boldsymbol{x}, \sigma^2)$ with fixed σ, then by conjugacy we can easily show that the posterior is also Gaussian

$$\begin{aligned} p(\boldsymbol{w}|\mathbf{X}, \boldsymbol{y}, \sigma^2) =& \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ \text{where} \quad \boldsymbol{\mu} = \frac{1}{\sigma^2} \boldsymbol{\Sigma} \mathbf{X}^\top \boldsymbol{y}&, \ \boldsymbol{\Sigma} = (I + \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X})^{-1}, \end{aligned} \tag{2.7}$$

using the same notation for $\mathbf{X}$ as Section 2.1.1. The corresponding posterior predictive is also Gaussian

$$\begin{aligned} p(y^*|\boldsymbol{x}^*, \mathbf{X}, \sigma^2) &= \int \mathcal{N}(y^*|\boldsymbol{w}^\top \boldsymbol{x}^*, \sigma^2) \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{w} \\ &= \mathcal{N}(y^*|\boldsymbol{\mu}^\top \boldsymbol{x}^*, \sigma^2 + \boldsymbol{x}^{*\top} \boldsymbol{\Sigma} \boldsymbol{x}). \end{aligned} \tag{2.8}$$

However inference is difficult in general due to the difficulty of computing the normalising constant in the denominator of Equation 2.4. The integral is usually intractable, so we need methods to approximate the posterior distribution or samples from it. The simplest approximation to the posterior is a point estimate, also known as *Maximum a posteriori*.

2.2.3 Maximum a Posteriori (MAP) and Empirical Bayes

The point estimate of the posterior is called the *Maximum a Posteriori (MAP)* estimate when it maximises the posterior density

$$\begin{aligned}\boldsymbol{\theta}^{MAP} &= \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{x}) \\ &= \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \end{aligned} \tag{2.9}$$

Then the predictive distribution can be approximated by $p(\boldsymbol{x}^*|\boldsymbol{\theta}^{MAP})$. This circumvents the need to compute the normalising constant of the posterior, since this constant is independent of $\boldsymbol{\theta}$. While being easy to compute, the obvious problem with this approach is that it does not give a measure of uncertainty about the latent, hence the predictive uncertainty tends to be underestimated. This can lead to overfitting the training data, and also to overconfident decision making that can be dangerous or undesirable at best. A more subtle disadvantage of MAP estimation is that it is not invariant to reparameterisation. Changing to an equivalent parameterisation can change the predictive distribution given by the MAP estimate, whereas exact inference doesn't face this problem – the change of measure is taken into account in the normalising constant when integrating over the parameter space to derive the posterior predictive.

A less simplistic approximation to the posterior is to fit a Gaussian to the posterior distribution, also known as *Laplace Approximation*. From the observation that the log density of a Gaussian is similar to the second order term of a Taylor series expansion, we use the Taylor expansion estimate of the negative log of the unnormalised posterior around the MAP estimate. Then we have

$$E(\boldsymbol{\theta}) \approx E(\boldsymbol{\theta}^{\mathrm{MAP}}) + (\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathrm{MAP}})^\top \nabla E(\boldsymbol{\theta})|_{\boldsymbol{\theta}^{\mathrm{MAP}}} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathrm{MAP}})^\top \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathrm{MAP}}), \tag{2.10}$$

where $E(\boldsymbol{\theta}) \coloneqq -\log p(\boldsymbol{x}, \boldsymbol{\theta})$, $\mathbf{H} \coloneqq \frac{\partial^2 E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^\top}|_{\boldsymbol{\theta}^{\mathrm{MAP}}}$.
By definition of $\boldsymbol{\theta}^{\mathrm{MAP}}$, $\nabla E(\boldsymbol{\theta})|_{\boldsymbol{\theta}^{\mathrm{MAP}}} = 0$. Hence

$$\begin{aligned} p(\boldsymbol{\theta}|\boldsymbol{x}) \approx& \frac{1}{p(\boldsymbol{x})} \exp(-E(\boldsymbol{\theta}^{\mathrm{MAP}})) \exp\Big[-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathrm{MAP}})^\top \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathrm{MAP}})\Big] \\ \propto& \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^{\mathrm{MAP}}, \mathbf{H}^{-1}). \end{aligned} \tag{2.11}$$

Hence the normalisation constant of this Gaussian distribution
$\exp(-E(\boldsymbol{\theta}^{\mathrm{MAP}}))(2\pi)^{D/2}|\mathbf{H}|^{-1/2}$ approximates the marginal likelihood $p(\boldsymbol{x})$.

MAP estimates can still be a useful baseline for models where the posterior is difficult to approximate, such as *hierarchical Bayesian* models. In such models, we put a further prior over the model hyperparameters, also known as *hyperpriors*. In other words, the parameter $\boldsymbol{\eta}$ of the prior $p(\boldsymbol{\theta}|\boldsymbol{\eta})$ has a further hyperprior $p(\boldsymbol{\eta})$. In this case, the full posterior distribution becomes

$$p(\boldsymbol{\theta}, \boldsymbol{\eta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\eta})p(\boldsymbol{\eta})}{\int p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\eta})p(\boldsymbol{\eta})d\boldsymbol{\theta}d\boldsymbol{\eta}}. \tag{2.12}$$

The added level of hierarchy usually makes exact inference even more difficult, since the integral (normalisation constant) is over both $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$, whereas MAP estimation of $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ remains simple.

A more Bayesian approach to inference in hierarchical models is called *empirical Bayes*. We use a point estimate for $\boldsymbol{\eta}$, but still perform exact inference for $\boldsymbol{\theta}$ given this point estimate of $\boldsymbol{\eta}$. In particular, we use $\hat{\boldsymbol{\eta}} \coloneqq \arg\max_{\boldsymbol{\eta}} p(\boldsymbol{\eta}|\boldsymbol{x}) = \arg\max_{\boldsymbol{\eta}} \int p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\eta})d\boldsymbol{\theta}$ i.e. the value that maximises the *marginal likelihood* of the data given $\boldsymbol{\eta}$. Hence this approach is also referred to as *type-II maximum likelihood*. This is often used for inference in the context of Gaussian Processes, described in Section 2.2.9 and also relevant for Chapter 4.

2.2.4 Monte Carlo Estimation

For cases where exact inference is difficult due to the intractability of the posterior, one workaround is to generate (approximate) samples from the posterior. Suppose we have a distribution π and we would like to evaluate $\mathbb{E}_{\pi}[f(\boldsymbol{\theta})]$ (let us assume that this expectation exists and is finite). If π is the prior, a corresponding example is the marginal likelihood $p(\boldsymbol{x}) = \mathbb{E}_{p(\boldsymbol{\theta})}[p(\boldsymbol{x}|\boldsymbol{\theta})]$. If π is the posterior, a corresponding example would be the posterior predictive distribution $\mathbb{E}_{p(\boldsymbol{\theta}|\boldsymbol{x})}[p(\boldsymbol{x}^*|\boldsymbol{\theta})]$. Then the *Monte Carlo (MC) estimate* of this quantity is $\bar{f^N} \coloneqq \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{\theta}^{(i)})$ where $\boldsymbol{\theta}^{(i)} \overset{\text{iid}}{\sim} \pi$. It is easy to see that this estimator is unbiased, and also by the Strong Law of Large Numbers, we can show that the estimator $\bar{f^N}$ converges almost surely to $\mathbb{E}_{\pi}[f(\boldsymbol{\theta})]$ as $N \to \infty$. Hence if we can sample from the posterior, then we can approximate this expectation arbitrarily well by taking arbitrarily many samples.

2.2.5 Importance Sampling

However the variance of the above MC estimator could be very large, meaning that we need N to be very large to get an accurate estimate. One method of addressing

this issue is *importance sampling*. Here we use an *importance distribution* q that we can sample from instead to form an alternative unbiased estimate. First note that

$$\begin{aligned}\mathbb{E}_\pi[f(\boldsymbol{\theta})] &= \int f(\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta} = \int \frac{f(\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}q(\boldsymbol{\theta})d\boldsymbol{\theta} \\ &= \mathbb{E}_q\Big[\frac{f(\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\Big]. \end{aligned} \tag{2.13}$$

Hence $\bar{f}_q^N \coloneqq \frac{1}{N}\sum_{i=1}^N \frac{f(\boldsymbol{\theta}^{(i)})\pi(\boldsymbol{\theta}^{(i)})}{q(\boldsymbol{\theta}^{(i)})}$ where $\boldsymbol{\theta}^{(i)} \overset{\text{iid}}{\sim} q$ is also an unbiased estimator. However here we need to know the value of the density π to evaluate the importance weights $w^{(i)} \coloneqq \frac{f(\boldsymbol{\theta}^{(i)})\pi(\boldsymbol{\theta}^{(i)})}{q(\boldsymbol{\theta}^{(i)})}$. It is straight-forward to show that the *optimal importance distribution*, the q that minimises the variance of the estimator, is $q^*(\boldsymbol{\theta}) = \frac{|f(\boldsymbol{\theta})|\pi(\boldsymbol{\theta})}{\int |f(\boldsymbol{\theta})|\pi(\boldsymbol{\theta})d\boldsymbol{\theta}}$. In the context of the marginal likelihood, the optimal importance distribution is the posterior $p(\boldsymbol{\theta}|\boldsymbol{x})$, and for the posterior predictive distribution it is the posterior given both $\boldsymbol{x}$ and $\boldsymbol{x}^*$, i.e. $p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{x}^*)$.

2.2.6 Markov Chain Monte Carlo (MCMC)

Returning to the MC estimate of $\mathbb{E}_\pi[f(\boldsymbol{\theta})]$, where π is the posterior, MCMC is a body of methods that sample from π via a Markov chain. Up to and including subsection 2.2.6, we adopt a change of notation to conform to standard notation used in the MCMC literature; we use X to denote random variables and x to denote their values. A Markov chain is a sequence of random variables $(X_n)_{n\in\mathbb{N}_0}$ that evolves in discrete time, such that the distribution of each state only depends on the previous state. In other words

$$\mathbb{P}(X_{n+1} = x_{n+1}|(X_i = x_i)_{i=0}^n) = \mathbb{P}(X_{n+1} = x_{n+1}|X_n = x_n) \;\; \forall x_i \in \mathcal{X}, n \in \mathbb{N}_0. \tag{2.14}$$

If the transition probabilities are time-independent, the Markov chain is said to be stationary, so the initial distribution μ over the state space $\mathcal{X} \in \mathbb{R}^d$ and the *transition kernel* $T(x \to x') \coloneqq \mathbb{P}(X_{n+1} = x'|X_n = x)$ fully define the distribution of the Markov chain. Let us assume for notational convenience that this transition kernel admits a probability density function p.

An important notion is *irreducibility*, which means that every $x \in \mathcal{X}$ has a non-zero probability of reaching any measurable subset of $\mathcal{X}$ that has non-zero measure via transitions of the Markov Chain. Looking at distributions over the states X_n and how these are affected by the transition kernel, we say that the Markov chain (X_n) has an *invariant* or *stationary* distribution π on $\mathcal{X}$ if $\int_\mathcal{X} \pi(x)p(x'|x)dx = \pi(x') \;\; \forall x' \in \mathcal{X}$ i.e.

the transition kernel leaves π unchanged. We say that π and T are in *detailed balance* if $\pi(x)p(x'|x) = \pi(x')p(x|x') \ \forall x, x' \in \mathcal{X}$. It can be interpreted as time reversibility. It is straightforward to show that detailed balance implies invariance, making detailed balance a useful property for defining invariant Markov chains, but the converse does not hold in general.

The principle of MCMC is to create a Markov chain whose stationary distribution π is the distribution we wish to sample from. Then the *Ergodic Theorem* tells us that assuming irreducibility and boundedness i.e. $\mathbb{E}_\pi[|f(X)|] < \infty$ for $f : \mathcal{X} \to \mathbb{R}$, we have $\frac{1}{N}\sum_{n=1}^{N} f(X_n) \overset{a.s.}{\to} \mathbb{E}_\pi[f(X)]$. We refer the reader to Meyn and Tweedie [2012] for a rigorous, measure-theoretic formulation of MCMC and proofs of ergodicity in discrete and continuous state space $\mathcal{X}$.

Metropolis Hastings (MH)

The *Metropolis Hastings* (MH) algorithm defines a Markov chain with stationary distribution π whose transition kernel we can define even when we only know π up to a normalising constant. Define $q(\cdot|x)$ to be the proposal distribution on $\mathcal{X}$ from which we can both sample and evaluate the density easily. Then the MH algorithm defines a Markov chain using proposals from q for the next state followed by an accept-reject step as follows

1. Choose $x_0 \in \mathcal{X}$ such that $\pi(x_0) > 0$ and set $X_0 = x_0$.
2. For $t = 0, \dots, T$:
   (a) generate $X'_t \sim q(\cdot|X_t)$ and $U_t \sim Unif[0, 1]$.
   (b) compute $\alpha(X_t, X'_t) = \min\left(\frac{\pi(X'_t)q(X_t|X'_t)}{\pi(X_t)q(X'_t|X_t)}, 1\right)$.
   (c) If $U_t \leq \alpha(X_t, X'_t)$, set $X_{t+1} = X'_t$ otherwise set $X_{t+1} = X_t$.

Since the transition kernel of MH only depends on π via the term $\pi(X'_t)/\pi(X_t)$, it suffices to know π up to a normalising constant. It can be shown that the MH algorithm generates a Markov chain with stationary distribution π that satisfies detailed balance. And if q is chosen such that irreducibility holds (e.g. if the support of q is $\mathcal{X}$), then ergodicity holds. Evidently, the choice of q has a big impact on how fast the MCMC chain reaches convergence. Some popular choices of q are the following

1. *Independent MH* $q(x'|x) = q(x') \Rightarrow \alpha(x, x') = \min\left(1, \frac{\pi(x')q(x)}{\pi(x)q(x')}\right)$.

2. *Metropolis Algorithm* $q(x'|x) = q(x|x') \Rightarrow \alpha(x, x') = \min\left(1, \frac{\pi(x')}{\pi(x)}\right)$.

3. *Random Walk MH* $q(x'|x) = q(|x' - x|) \Rightarrow \alpha(x, x') = \min\left(1, \frac{\pi(x')}{\pi(x)}\right)$.

Gibbs Sampling

The design of the proposal distribution q for MH to ensure a high enough acceptance probability as well as fast mixing can pose a major challenge. An alternative sampling algorithm that does not require the choice of a proposal is *Gibbs sampling*. This MCMC algorithm is suitable when we can sample from the conditional distributions of the target distribution, namely $\pi(x_i|x_{-i})$, where $x_{-i} \coloneqq (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p)$. Gibbs sampling amounts to cycling through the coordinates of x, updating x by sampling from each conditional distribution until convergence as follows

1. Choose $x_0 \in \mathcal{X} \subset \mathbb{R}^p$ such that $\pi(x_0) > 0$ and set $X_0 = x_0$.

2. For $t = 0, \ldots, T$:

   (a) For $i = 1, \ldots, p$:

      i. $X_{t,i} \sim \pi(x_i|x_{-i} = X_{t,-i})$.

Gibbs sampling is used, for instance, for inference in Bayesian matrix factorisation models in Section 2.2.10. Although Gibbs sampling is an attractive method when these conditional distributions can be obtained analytically, they can often suffer from slow convergence to the target distribution. This is particularly a problem when x is high dimensional, and the dimensions are highly correlated under the stationary distribution.

Practicalities of MCMC: Implementation and Convergence Diagnostics

In practice, the MCMC chain starts from some fixed state, and it takes multiple transitions for the chain to approach the stationary distribution. This initial phase is called the *burn-in* or *warm-up* phase, and the samples during this phase are discarded. A conservative rule of thumb is to discard the first half of the simulation [Gelman et al., 2013]. It is also standard practice to use samples from multiple chains with starting points drawn from an over-dispersed distribution, and investigate the between-chain and within-chain behaviour to assess convergence. The chains should *mix*, in that

different chain trajectories should trace out a common distribution, and each individual chain should reach stationarity. For each scalar parameter θ with samples labelled θ_{ij} for $i = 1, \ldots, n,\ \ j = 1, \ldots, m$ (m chains each of length n), the key quantities for monitoring such behaviour are B and W, the between- and within-chain variances

$$B = \frac{n}{m-1} \sum_{j=1}^{m} (\bar{\theta}_{\cdot j} - \bar{\theta}_{\cdot\cdot})^2, \ \text{ where } \bar{\theta}_{\cdot j} = \frac{1}{n} \sum_{i=1}^{n} \theta_{ij}, \ \ \bar{\theta}_{\cdot\cdot} = \frac{1}{m} \sum_{j=1}^{m} \bar{\theta}_{\cdot j}$$

$$W = \frac{1}{m} \sum_{j=1}^{m} s_j^2, \ \text{ where } s_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\theta_{ij} - \bar{\theta}_{\cdot j})^2. \tag{2.15}$$

$$\tag{2.16}$$

The estimate for the posterior variance $\mathrm{Var}(\theta)$ is a weighted average of W and B, namely $\widehat{\mathrm{Var}}(\theta) \coloneqq \frac{n-1}{n} W + \frac{1}{n} B$, which is unbiased under stationarity. Until the chain mixes, the within-chain variance estimate W will be an underestimate of $\mathrm{Var}(\theta)$ since each chain will have less variability than between chains. Hence a natural quantity to monitor is the *Gelman-Rubin Diagnostic*, also known as the *R-hat*, the ratio of the estimated posterior variance and the within-chain variance. This quantity should decline to 1 upon mixing

$$\hat{R} = \sqrt{\frac{\widehat{\mathrm{Var}}(\theta)}{W}}. \tag{2.17}$$

Another heuristic for convergence is to look at the *effective sample size* (ESS), which estimates how many *iid* samples have the same estimation performance of the posterior mean $\mathbb{E}(\theta)$ as the correlated samples of the MCMC chain. Since $\bar{\theta}_{\cdot\cdot}$ is the chain estimate of the posterior mean, we look at $\mathrm{Var}(\theta)/\mathrm{Var}(\bar{\theta}_{\cdot\cdot})$, which is equal to the total number of samples, mn, at convergence. To estimate this quantity, we can use the asymptotic formula for the variance of the average of a correlated sequence

$$\lim_{n \to \infty} mn \mathrm{Var}(\bar{\theta}_{\cdot\cdot}) = \Big(1 + 2 \sum_{t=1}^{\infty} \rho_t\Big) \mathrm{Var}(\theta) \tag{2.18}$$

where ρ_t is the autocorrelation of the sequence at lag t. This is estimated using the formula $\rho_t = 1 - \frac{\mathbb{E}[(X_i - X_{i-t})^2]}{2 \mathrm{Var} X_i}$ for sequence (X_i)

$$\hat{\rho}_t = 1 - \frac{1}{2\hat{\mathrm{Var}}(\theta)} \frac{1}{m(n-t)} \sum_{j=1}^{m} \sum_{i=t+1}^{n} (\theta_{i,j} - \theta_{i-t,j})^2. \tag{2.19}$$

The $\hat{\rho}_t$ are summed up until time T where the sum of the autocorrelation for two successive lags is negative. Hence the ESS is given by

$$\mathrm{ESS} = \frac{mn}{1 + 2 \sum_{t=1}^{T} \hat{\rho}_t}. \tag{2.20}$$

However ESS can be misleading since it can be very high even when between-chain variance is high i.e. when the chain has not mixed. Hence it should only be looked at when the value of *R-hat* is close to 1. All these quantities are computed with samples after discarding those from the burn-in phase. In Chapter 3, we implement MCMC as above for inference, using these diagnostics to assess convergence.

Hamiltonian Monte Carlo (HMC)

One of the reasons an MCMC chain can show slow mixing is that it can get stuck in a mode of the stationary distribution. Hence we need to be able to suggest distant proposals via a suitable choice of q in the MH algorithm. A popular go-to algorithm for such distant proposals is *Hamiltonian Monte Carlo* (HMC) [Neal, 2011]. The core idea is to augment the parameter space with a space for the momentum variable of Hamiltonian dynamics, form a new target density on this augmented space whose marginal density is the original unnormalised posterior, then run MH on the augmented space using Hamiltonian dynamics for generating proposals.

In Hamiltonian dynamics, the state of an object in a physical system is described by its position $\boldsymbol{\theta}$ and momentum $\boldsymbol{\phi}$. The *potential energy* $U(\boldsymbol{\theta})$ is a function of position only, whereas the *kinetic energy* $K(\boldsymbol{\phi})$ is a function of momentum only. The system is described by the *Hamiltonian* $H(\boldsymbol{\theta}, \boldsymbol{\phi})$ where $\boldsymbol{\theta}, \boldsymbol{\phi}$ change with time t according to *Hamilton's equations* $\frac{d\phi_i}{dt} = \frac{\partial H}{\partial \theta_i}$, $\frac{d\theta_i}{dt} = -\frac{\partial H}{\partial \phi_i}$. The Hamiltonian has the following notable properties.

1. *Conservation of Hamiltonian.* It can easily be derived that $\frac{dH}{dt} = 0$.
2. *Reversibility.* The mapping $T_s : (\boldsymbol{\theta}(t), \boldsymbol{\phi}(t)) \mapsto (\boldsymbol{\theta}(t+s), \boldsymbol{\phi}(t+s))$ is bijective, so has an inverse T_{-s}. Assuming $H(\boldsymbol{\theta}, \boldsymbol{\phi}) = U(\boldsymbol{\theta}) + K(\boldsymbol{\phi})$ and $K(\boldsymbol{\phi}) = K(-\boldsymbol{\phi})$, T_{-s} can be obtained by negating $\boldsymbol{\phi}$, applying T_s, then negating $\boldsymbol{\phi}$ again.
3. *Volume preservation.* $T_s(R)$ has the same volume as R, where R is a subset of the augmented $(\boldsymbol{\theta}, \boldsymbol{\phi})$ space.

Take $H(\boldsymbol{\theta}, \boldsymbol{\phi}) = U(\boldsymbol{\theta}) + K(\boldsymbol{\phi})$ with $U(\boldsymbol{\theta}) = -\log[p(\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta})]$, $K(\boldsymbol{\phi}) = \frac{1}{2}\boldsymbol{\phi}^\top M^{-1}\boldsymbol{\phi}$, $M = diag(m_1, \ldots, m_d)$, interpreting $\exp(-H(\boldsymbol{\theta}, \boldsymbol{\phi}))$ as the unnormalised target density of $(\boldsymbol{\theta}, \boldsymbol{\phi})$. To simulate Hamiltonian dynamics in practice, Hamiltonian's equations must be discretised. One particular method of discretisation is the called the *Leapfrog*

method. For step size ϵ, Hamilton's equations become

$$\begin{aligned}
\phi_i(t+\frac{\epsilon}{2}) =& \phi_i(t) - \frac{\epsilon}{2}\frac{\partial U(\boldsymbol{\theta}(t))}{\partial \theta_i} \quad \text{(half step for momentum)} \\
\theta_i(t+\epsilon) =& \theta_i(t) + \epsilon\frac{\phi_i(t+\epsilon/2))}{m_i} \quad \text{(full step for position)} \\
\phi_i(t+\epsilon) =& \phi_i(t+\frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{\partial U(\boldsymbol{\theta}(t+\epsilon))}{\partial \theta_i} \quad \text{(further half step for momentum).}
\end{aligned}$$

The leapfrog method still has reversibility and preserves volume, but conservation of the Hamiltonian does not hold due to discretisation error. Given some initial $\boldsymbol{\theta}$, the HMC algorithm runs as follows

1. Sample $\boldsymbol{\phi} \sim p(\boldsymbol{\phi}) \propto \exp(-K(\boldsymbol{\phi}))$ (Gaussian with variance given by mass matrix M).
2. Metropolis update using Hamiltonian dynamics for proposing a new state:
   (a) Run L steps of the leapfrog method with step size ϵ, to obtain proposed state $(\boldsymbol{\theta}^*, \boldsymbol{\phi}^*)$.
   (b) *Accept-reject*. Set $(\boldsymbol{\theta}, \boldsymbol{\phi}) = (\boldsymbol{\theta}^*, \boldsymbol{\phi}^*)$ with probability $\min\left(1, \frac{\exp(-H(\boldsymbol{\theta}^*, \boldsymbol{\phi}^*))}{\exp(-H(\boldsymbol{\theta}, \boldsymbol{\phi}))}\right)$.

The first step can change the joint density by a large value, so this is what allows us to make big jumps in the MCMC chain. Note that reversibility does not necessarily imply symmetry of the marginal proposal distribution of $\boldsymbol{\theta}$. This is because the proposal distribution $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}) = \int \delta_{T_L(\boldsymbol{\theta},\boldsymbol{\phi})}(\boldsymbol{\theta}^*)p(\boldsymbol{\phi})$ is obtained by integrating out the momentum distribution, where $T_{\epsilon L}$ represents L steps of size ϵ the leapfrog method, a deterministic mapping. Tierney et al. [1998] shows that the property of volume preservation, together with reversibility, ensures symmetry of the proposal. Consequently HMC is an instance of the *Metropolis Algorithm* above, hence there is no need to be able to compute the density of the proposal in order to compute the acceptance probability.

In practice, HMC requires tuning of the number of steps L and step size ϵ for fast convergence. The *No-U-Turn Sampler* (NUTS) [Hoffman and Gelman, 2014] modifies HMC to eliminate the need to choose L, and also provides a scheme for automatically tuning ϵ. A high level description of NUTS is that it uses a U-turn criterion to tell us when we have simulated for "long enough"; it uses the distance between the proposal $\boldsymbol{\theta}^*$ and the initial value of $\boldsymbol{\theta}$, and stops running more leapfrog steps when the proposed momentum $\boldsymbol{\phi}^*$ points in the opposite direction of $\boldsymbol{\theta}^* - \boldsymbol{\theta}$. However this alone does not guarantee reversibility as in vanilla HMC, hence there are extra steps involved to

make sure the process of generating the proposal is reversible. Further details can be found in Hoffman and Gelman [2014].

2.2.7 Variational Inference (VI)

Although MCMC guarantees sampling from the true posterior distribution upon convergence, it can be difficult to reach convergence and there only exist proxies to assess convergence. Moreover, MCMC typically requires looking at all of the data per transition of the Markov chain, making the computational cost scale at $O(N)$ for every step of Metropolis Hastings, for example. Using a stochastic method that can run on mini-batches of data while preserving convergence guarantees is difficult, although there exist literature under the heading of *stochastic MCMC* (e.g. Chen et al. [2014]). A more computationally efficient alternative (that may sacrifice accuracy of inference) is *variational inference* (VI). The basic idea of VI is to choose a family of distributions $\mathcal{F}$, the *variational family*, and compute the *variational distribution* $q \in \mathcal{F}$ that is closest to the posterior distribution p in terms of reverse-KL distance

$$q^* = \arg\min_{q \in \mathcal{F}} D_{\mathrm{KL}}(q||p). \tag{2.21}$$

This approach converts the sampling problem into an optimisation problem. The difficulty here is that $\mathcal{F}$ must be chosen to be wide enough so that q^* is a reasonable approximation to p, but if $\mathcal{F}$ is too wide then optimisation is usually difficult. The key is to find a family that gives a good trade-off between flexibility and ease of optimisation. Switching to canonical notation in VI, so that $\boldsymbol{z}$ is the latent variable for inference, note that

$$\begin{aligned} D_{\mathrm{KL}}(q||p) =& \mathbb{E}_{q(\boldsymbol{z})}\Big[\log \frac{q(\boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x})}\Big] \\ =& \mathbb{E}_{q(\boldsymbol{z})}\Big[\log \frac{q(\boldsymbol{z})p(\boldsymbol{x})}{p(\boldsymbol{z},\boldsymbol{x})}\Big] \\ =& \mathbb{E}_{q(\boldsymbol{z})}\Big[\log \frac{q(\boldsymbol{z})}{p(\boldsymbol{z},\boldsymbol{x})}\Big] + \log p(\boldsymbol{x}) \geq 0. \end{aligned} \tag{2.22}$$

From this decomposition, we can see that $q^* = \arg\min_{q \in \mathcal{F}} D_{\mathrm{KL}}(q||p)$ is equivalent to

$$q^* = \arg\max_{q \in \mathcal{F}} \mathbb{E}_{q(\boldsymbol{z})}\Big[\log \frac{p(\boldsymbol{z},\boldsymbol{x})}{q(\boldsymbol{z})}\Big]. \tag{2.23}$$

The objective on the right hand side is also known as the *evidence lower bound* (ELBO), since it is a lower bound on the marginal log likelihood $\log p(\boldsymbol{x})$. Note that the ELBO

can also be rewritten as

$$\mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{x}|\boldsymbol{z})] - D_{\mathrm{KL}}(q(\boldsymbol{z})||p(\boldsymbol{z})), \tag{2.24}$$

which may be easier to estimate than the original formulation if the $D_{\mathrm{KL}}(q(\boldsymbol{z})||p(\boldsymbol{z}))$ term is analytic. Although the reverse-KL provides a computationally convenient method for obtaining the variational distribution, alternative divergences can also be used, such as the α-divergence (also known as Rényi divergence) [Minka, 2005, Li and Turner, 2016] of which the reverse-KL is a special case.

Variational Learning from the Max-Max View of EM

If we were to learn the model p as well as q, the ELBO also happens to be a sensible objective with respect to p. This is very closely related to the *Expectation Maximisation* (EM) algorithm. In EM, we have a latent variable model $p_\theta(\boldsymbol{x}, \boldsymbol{z})$, where the subscript indicates that the density p is parameterised by θ. The goal is to optimise θ so as to maximise the marginal likelihood $p_\theta(\boldsymbol{x})$. EM achieves this by iterating over two steps

1. *E-step.* $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) = \mathbb{E}_{p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})]$.
2. *M-step.* $\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})$.

Assuming that the expectation and the maximal $\boldsymbol{\theta}$ can be computed exactly, the EM algorithm is guaranteed to converge. *Generalised EM* relaxes the M-step to a partial maximisation, and convergence is still guaranteed. However the difficulty usually lies in the E-step that requires computing exact expectations with respect to the posterior, which is only plausible in limited settings. Noting that the posterior is the q^* that maximises the ELBO, we can reformulate EM to the maximisation-maximisation view allowing us to see how the E-step can also be relaxed [Neal and Hinton, 1998]

1. *E-step.* $q^{(t)} = \arg\max_q \mathbb{E}_{q(\boldsymbol{z})}\left[\log \frac{p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})}\right] = p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{z}|\boldsymbol{x})$.
2. *M-step.* $\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{q^{(t)}(\boldsymbol{z})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q^{(t)}(\boldsymbol{z})}\right]$.

This reformulation of the EM justifies partial q updates in the E-step, in other words updating q to increase the ELBO without having to find the optimal q.

Mean-Field VI

A popular choice for the variational family is the set of fully factorised q, and this choice leads to *mean-field* VI. Here one assumes $q(\boldsymbol{z}) = \prod_j q(z_j)$, and solves the problem $\min_{q_1,\ldots,q_D} D_{\mathrm{KL}}(q||p)$. Singling out the terms of the ELBO that involve q_j, it is easy to see that the objective for q_j simplifies to

$$\begin{aligned} q_j^*(z_j) &= \arg\max_{q_j} \mathbb{E}_{q_j(z_j)}\left[\mathbb{E}_{q_{-j}(\boldsymbol{z}_{-j})}[\log p(\boldsymbol{x}, \boldsymbol{z})] - \log q_j(z_j)\right] \\ &= \arg\max_{q_j} -D_{\mathrm{KL}}(q_j||r_j), \end{aligned} \tag{2.25}$$

where $r_j \coloneqq \mathbb{E}_{q_{-j}(\boldsymbol{z}_{-j})}[\log p(\boldsymbol{x}, \boldsymbol{z})]$. Hence

$$q_j^*(z_j) = \mathbb{E}_{q_{-j}(\boldsymbol{z}_{-j})}[\log p(\boldsymbol{x}, \boldsymbol{z})]. \tag{2.26}$$

If $q_j^*(z_j)$ is intractable, then standard practice is to optimise the objective in Equation 2.25 with partial maximisation steps.

Stochastic VI

Stochastic variational inference [Hoffman et al., 2013] extends the scope of variational inference to massive datasets. It is used in the context of local latent variable models, where each data point $\boldsymbol{x}^{(i)}$ only depends on a designated latent variable $\boldsymbol{z}^{(i)}$ and is independent of the other data points or local latents. The method is applied to conjugate models whose priors and likelihoods are from the exponential family with mean-field variational posteriors, making them suitable for optimising the ELBO with *natural gradients* [Amari, 1998] with respect to q. Natural gradient descent can be thought of as a corrected version of gradient descent when optimising an objective with respect to a parameterised distribution. While gradient descent gives the direction of fastest change of the objective with respect to Euclidean distance between the parameters, natural gradient descent gives the direction of fastest change with respect to KL divergence between the distributions. Applied to conjugate models, natural gradients simplify greatly and give simple update rules. The other contribution of stochastic variational inference is that each q update only depends on a mini-batch of data as opposed to the whole dataset (the case for classic variational inference). In

other words, we rewrite the ELBO as follows

$$\begin{aligned}\log p(\boldsymbol{x}) &= \sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)}) \\ &= \sum_{i=1}^{N} \mathbb{E}_{q(\boldsymbol{z}^{(i)})}\Big[\log \frac{p(\boldsymbol{z}^{(i)}, \boldsymbol{x}^{(i)})}{q(\boldsymbol{z}^{(i)})}\Big] \qquad (2.27) \\ &\approx \sum_{i\in\mathcal{B}} \mathbb{E}_{q(\boldsymbol{z}^{(i)})}\Big[\log \frac{p(\boldsymbol{z}^{(i)}, \boldsymbol{x}^{(i)})}{q(\boldsymbol{z}^{(i)})}\Big]. \qquad (2.28)\end{aligned}$$

This stochastic approximation of the dataset using a single mini-batch (that is randomly sampled at each training iteration) is what allows stochastic VI to scale to massive datasets, reducing per-step computational complexity from $O(N)$ to $O(|\mathcal{B}|)$.

Black Box VI (BBVI)

The mean-field updates above are usually only applicable to a restricted class of models, and the update rules have to be derived separately for each model. To extend the scope of variational inference to a larger class of models, there has been work on *black box VI* (BBVI) that gives generally applicable inference procedures. The idea is to devise a $q_{\boldsymbol{\phi}}$ whose density is continuous in its parameters $\boldsymbol{\phi}$, and use the score function estimator [Fu, 2006], also known as REINFORCE [Williams, 1992] or likelihood-ratio estimator [Glynn, 1990] for the gradients with respect to $\boldsymbol{\phi}$. Specifically, it uses the identity $\nabla_{\boldsymbol{\phi}} q_{\boldsymbol{\phi}}(\boldsymbol{z}) = q_{\boldsymbol{\phi}}(\boldsymbol{z}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z})$, from which we have

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[f(\boldsymbol{z})] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[f(\boldsymbol{z}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z})] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[\nabla_{\boldsymbol{\phi}} f(\boldsymbol{z})] \qquad (2.29)$$

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[\nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z})] = \int \nabla_{\boldsymbol{\phi}} q_{\boldsymbol{\phi}}(\boldsymbol{z}) d\boldsymbol{z} = \nabla_{\boldsymbol{\phi}} \int q_{\boldsymbol{\phi}}(\boldsymbol{z}) d\boldsymbol{z} = \nabla_{\boldsymbol{\phi}} 1 = 0. \qquad (2.30)$$

This allows the gradient of the ELBO to be written as an expectation, which is then estimated using Monte Carlo

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}\Big[\log \frac{p(\boldsymbol{z}, \boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z})}\Big] &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}\Big[\log \frac{p(\boldsymbol{z}, \boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z})} \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z})\Big] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[\nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z})] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}\Big[\log \frac{p(\boldsymbol{z}, \boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z})} \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z})\Big] \qquad (2.31) \\ &\approx \frac{1}{S} \sum_{i=1}^{S} \log \frac{p(\boldsymbol{z}^{(i)}, \boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}^{(i)})} \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}^{(i)}) \text{ where } \boldsymbol{z}^{(i)} \overset{iid}{\sim} q_{\boldsymbol{\phi}}. \qquad (2.32)\end{aligned}$$

However this gradient estimate is known to have high variance, and a standard technique to reduce its variance is to add a baseline to the $\log \frac{p(\boldsymbol{z}^{(i)}, \boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}^{(i)})}$ term that

is independent of $\boldsymbol{z}^{(i)}$, lowering the magnitude of gradient estimates by exploiting Equation 2.30. There have been a plethora of different designs for these baselines [Paisley et al., 2012, Ranganath et al., 2014, Gregor et al., 2013, Mnih and Gregor, 2014, Titsias and Lázaro-Gredilla, 2014, Mnih and Rezende, 2016b].

Variational Autoencoder (VAE)

Variational Autoencoders (VAEs) [Kingma and Welling, 2013] combine various ideas in variational inference and learning to train a local latent variable model with deep neural network layers that map the local latent $\boldsymbol{z}^{(i)}$ to a data point $\boldsymbol{x}^{(i)}$. It uses stochastic gradient descent with two new ideas: *amortised* VI and the *reparameterisation trick*.

Amortised VI indicates that the variational posterior $q(\boldsymbol{z}^{(i)})$ for each local latent is parameterised as a function of $\boldsymbol{x}^{(i)}$, with shared parameters across all data points. This is usually written as $q_{\boldsymbol{\phi}}(\boldsymbol{z}^{(i)}|\boldsymbol{x}^{(i)})$. This is in contrast to classical VI for local latent variable models, where the $(q(\boldsymbol{z}^{(i)}))_{i=1}^N$ are separate distributions that are optimised independently of one another, for example in Stochastic VI. An example choice is $q_{\boldsymbol{\phi}}(\boldsymbol{z}^{(i)}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{z}^{(i)}|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}), \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}))$, a Gaussian whose mean and variance are derived from the output of a deep neural network with input $\boldsymbol{x}^{(i)}$ and weights $\boldsymbol{\phi}$. Hence this deep neural net is usually referred to as an *inference network*. Amortisation has a number of benefits: since it only requires storing the $\boldsymbol{\phi}$, it is applicable even when the number of data points is very large, when it is infeasible to store the variational parameters of each $q(\boldsymbol{z}^{(i)})$ in memory. Moreover we would expect similar data points to have similar posteriors, which is well-reflected in the amortised setup as inductive bias. Finally, the *inference network* is readily applicable to unseen test data, a feature that is not available for classical VI.

The *reparameterisation trick* [Kingma and Welling, 2013, Rezende et al., 2014, Titsias and Lázaro-Gredilla, 2014] gives gradient estimates that have much lower variance than those used for BBVI for *reparameterisable* variational distributions. These distributions are reparameterisable in the sense that the samples of $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ can be written as a differentiable function $g_{\boldsymbol{\phi}}$ of the input $\boldsymbol{x}^{(i)}$ and independent noise $\boldsymbol{\epsilon}$ i.e. $z = g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x}^{(i)})$. Then expectations with respect to $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ can be estimated with an unbiased estimator

$$\begin{aligned}\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}[f(\boldsymbol{z})] =& \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x}^{(i)}))] \\ \approx& \frac{1}{L}\sum_{l=1}^{L} f(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}^{(i)})) \text{ where } \boldsymbol{\epsilon}^{(l)} \overset{iid}{\sim} p(\boldsymbol{\epsilon}).\end{aligned} \tag{2.33}$$

The location-scale family of distributions is an example of a reparameterisable distribution. For example a sample from a diagonal Gaussian $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}), \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}))$ is equivalent to $\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x}^{(i)}) \coloneqq \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}) \odot \boldsymbol{\epsilon}$, where $\odot$ is elementwise multiplication and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$. When the ELBO is an expectation with respect to a reparameterisable q, the main idea of the reparameterisation trick is to form a Monte Carlo estimate of the expectation, then take the gradient of this estimate using the reparameterised form to be the estimate of the gradient of the expectation. Hence the unbiased, estimated gradient of the ELBO is

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})} \Big[\log \frac{p(\boldsymbol{z}, \boldsymbol{x}^{(i)})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})} \Big] \approx \frac{1}{L} \sum_{l=1}^{L} \nabla_{\boldsymbol{\phi}} \log \frac{p(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}^{(i)}), \boldsymbol{x}^{(i)})}{q_{\boldsymbol{\phi}}(g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}^{(i)})|\boldsymbol{x}^{(i)})}. \tag{2.34}$$

Using the alternative formulation of the ELBO in Equation 2.24, we can form an alternative unbiased estimate

$$\frac{1}{L} \sum_{l=1}^{L} [\nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{x}^{(i)}|g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}^{(i)}))] - \nabla_{\boldsymbol{\phi}} D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p(\boldsymbol{z})). \tag{2.35}$$

Both estimates above are shown to have much lower variance than the score function gradient estimate in Equation 2.32 [Kingma and Welling, 2013], hence the reparameterisation trick can be a useful tool for BBVI. However note that the trick is limited to distributions that are reparameterisable, a subset of continuous distributions. Hence the reparameterisation trick is not applicable to discrete latent variables, for example. There exist, however, methods that use continuous approximations of discrete latent variables for applying the reparameterisation trick to discrete distributions [Maddison et al., 2016, Jang et al., 2016].

The inference network of VAEs can be thought of as a probabilistic encoder that maps data points to a distribution of latent variables. Interpreting the latent variables as representations of the data VAEs can be a useful method for learning abstract representations of the data. This observation provides the basis of Chapter 5.

2.2.8 Bayesian Model Selection

Given multiple models that fit a given dataset, it is natural to ask which model fits the data best. A standard approach to model selection in machine learning is to choose the model that shows the best performance on the metric of interest e.g. generalisation performance on validation/test data. When such data is not available, we may use K-fold *cross validation*. That is, we split the data randomly into K equally sized

partitions, and fit the model to the union of $K-1$ partitions K times, leaving out a different partition each time. Then we can look at the metric of interest of each trained model on the remaining partition, and compare the mean of these metric values across different models.

When there is no clear metric of interest, a Bayesian approach to model selection is to compute the posterior over models

$$p(m|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|m)p(m)}{\int p(\boldsymbol{x}|m)p(m)dm}, \tag{2.36}$$

where $\boldsymbol{x}$ is the dataset, and $p(m)$ denotes a prior over the space of models. We then select the MAP model by choosing the value of m that maximises the above posterior. Should we use a uniform prior over models, this amounts to choosing the model with the highest marginal likelihood (evidence) $p(\boldsymbol{x}|m)$. One might think that complex models with a large number of parameters is likely to have high likelihood. This is true for point estimates of the parameters that maximise the likelihood (e.g. MLE/MAP), but this is not necessarily true for the marginal likelihood, that measures likelihood integrating out the parameters over their prior distribution. This observation is referred to as *Bayesian Occam's razor* [MacKay, 1992], named after the principle *Occam's razor*, that one should pick the simplest model (in terms of model complexity) among those that fit the data well.

An intuitive explanation for Bayesian Occam's razor is as follows: note that the marginal likelihood over all possible datasets sums to 1 i.e. $\int p(\boldsymbol{x}|m)d\boldsymbol{x} = 1$ where the integral is over all possible datasets. Complex models with a high number of parameters will spread its probability mass over a wider region in the dataset space, because they can fit a wider range of datasets. Hence the marginal likelihood for any particular dataset is likely to be lower than a less complex model (whose mass is distributed over a smaller region) that can also fit the dataset. This reasoning does not hold for likelihoods with point estimates of the parameters $\hat{\boldsymbol{\theta}}$, since these parameters are a function of the dataset i.e. $\hat{\boldsymbol{\theta}}(\boldsymbol{x})$. Thus the conservation of mass does not necessarily hold upon integrating over all possible datasets i.e. $\int p(\boldsymbol{x}|m_{\hat{\boldsymbol{\theta}}(\boldsymbol{x})})d\boldsymbol{x} \neq 1$.

However the computation of the marginal likelihood is usually only tractable for conjugate models, and is intractable in general. In this case, we can opt to use an approximation to the log marginal likelihood that only requires the MLE $\hat{\boldsymbol{\theta}}$, known as the *Bayesian Information Criterion* (BIC)

$$BIC \coloneqq \log p(\boldsymbol{x}|\hat{\boldsymbol{\theta}}) - \frac{p}{2}\log N, \tag{2.37}$$

where p is the number of free parameters in $\boldsymbol{\theta}$ (so for real parameters, $\boldsymbol{\theta} \in \mathbb{R}^p$), and N is the number of data points in $\boldsymbol{x}$. This can be interpreted as a penalised marginal likelihood, where the penalty is linear in the number of free parameters in the model. The BIC can be derived by a Laplace approximation (c.f. Equation 2.10) to the marginal likelihood and the weak law of large numbers, with a detailed derivation in Schwarz et al. [1978]. An important side note is that the BIC can also be approximated with the MAP estimate instead of the MLE, especially for models where the MLE is poorly behaved [Fraley and Raftery, 2007].

In the case of hierarchical Bayesian models, where we use empirical Bayes (c.f. Section 2.2.3) to compute the MAP estimates of hyperparameters $\boldsymbol{\eta}$ while integrating out the parameters $\boldsymbol{\theta}$, the BIC score is

$$BIC \coloneqq \log p(\boldsymbol{x}|\hat{\boldsymbol{\eta}}) - \frac{P}{2} \log N, \tag{2.38}$$

where P is the number of free parameters in $\boldsymbol{\eta}$ and $p(\boldsymbol{x}|\hat{\boldsymbol{\eta}})$ is the marginal likelihood (integrating out $\boldsymbol{\theta}$) evaluated at the hyperparameter MAP estimate $\hat{\boldsymbol{\eta}}$. This is used throughout Chapter 4 for model selection.

2.2.9 Gaussian Processes

In the following sections, we will introduce some examples of probabilistic models that are used in Bayesian machine learning. One notable example is a *Gaussian Process* (GP) [Rasmussen and Williams, 2005], a model typically used for supervised learning, that forms the basis of Chapters 3 and 4. A Gaussian Process is a (infinite) collection of random variables, any finite number of which have consistent (defined below) Gaussian distributions. It is a particular example of a *stochastic process*. We denote the GP as $\boldsymbol{f} \sim \mathcal{GP}(m, k)$ over domain $\mathcal{X}$, where $m(\boldsymbol{x}) \in \mathbb{R}$ is the mean function and $k(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}$ is the covariance function for $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$, also referred to as the kernel (c.f. Section 2.1.2). Given inputs $(\boldsymbol{x}^{(i)})_{i=1}^N$, define $f_i \coloneqq \boldsymbol{f}(\boldsymbol{x}^{(i)})$ for $i \in \mathcal{I}$. Since we have a random variable for every possible value of $\boldsymbol{x} \in \mathcal{X}$, the GP can be thought of as a random function on the domain $\mathcal{X}$. The mean and variance of the finite dimensional marginals of the GP evaluated at these data points are given by these mean and covariance functions as follows:

$$\boldsymbol{f}_I \sim \mathcal{N}(\boldsymbol{\mu}_I, \mathbf{K}_{II}), \tag{2.39}$$

where $\boldsymbol{\mu}_I \coloneqq m(\boldsymbol{x}^{(i)})_{i \in I}$, $\mathbf{K}_{II} \coloneqq k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})_{i,j \in I}$ and I a finite subset of $\mathcal{I}$.

$\boldsymbol{f}$ is said to be *consistent* if it satisfies the marginalisation property

$$(\boldsymbol{f}_{I\cup J}) \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}) \Rightarrow \boldsymbol{f}_I \sim \mathcal{N}(\boldsymbol{\mu}_I, \mathbf{K}_{II}), \tag{2.40}$$

for arbitrary finite index sets $I, J \subset \mathcal{I}$. For GPs, it can easily be shown that $\boldsymbol{f}$ is consistent if and only if k is symmetric and positive definite, in other words, if k is a Mercer kernel (c.f. Section 2.1.2) [Rasmussen and Williams, 2005].

We may draw samples from a GP at the points $(\boldsymbol{x}^{(i)})_{i\in I}$ via joint generation, where we draw $\boldsymbol{f}_I \sim \mathcal{N}(\boldsymbol{\mu}_I, \mathbf{K}_{II})$. This requires R, the Cholesky decomposition of $\mathbf{K}_{II}$ i.e. $RR^\top = \mathbf{K}_{II}$, whose computation has a time complexity of $O(N^3)$ for $N \coloneqq |I|$. Then $R\boldsymbol{z} + \boldsymbol{\mu}_I$ for $\boldsymbol{z} \sim \mathcal{N}(0, I)$ is a sample from the GP at $\boldsymbol{x}_I$. In practice, we may need to add a small diagonal matrix to $\mathbf{K}_{II}$ to ensure that it is non-singular given floating point error and that its Cholesky decomposition computation is stable.

GPs for Regression

GPs are useful models for regression where the goal is to predict outputs $\boldsymbol{y} \coloneqq (y^{(i)})_{i=1}^N$ from inputs $X \coloneqq (\boldsymbol{x}^{(i)})_{i=1}^N$. A standard model for GP regression is a Gaussian likelihood $\boldsymbol{y}|X, \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}(X), \sigma_n^2 I)$ and a zero-mean GP prior $\boldsymbol{f} \sim \mathcal{GP}(0, k)$ where $\boldsymbol{f}(X)_i = \boldsymbol{f}(\boldsymbol{x}^{(i)})$ and σ_n is the standard deviation of observation/likelihood noise. Note that we usually assume $m \coloneqq 0$ for simplicity, and this assumption is not limiting since the posterior mean is not necessarily zero. For a new data point $\boldsymbol{x}^*$, noting that the joint distribution of $(\boldsymbol{y}, f(\boldsymbol{x}^*))$ is Gaussian, we may derive $\boldsymbol{f}(\boldsymbol{x}^*)|X, \boldsymbol{y}$, the conditional distribution of this joint

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}^*)|X, \boldsymbol{y} \sim \mathcal{N}(&k(X, \boldsymbol{x}^*)^\top [k(X, X) + \sigma_n^2 I]^{-1} \boldsymbol{y}, \\ &k(\boldsymbol{x}^*, \boldsymbol{x}^*) - k(X, \boldsymbol{x}^*)^\top [k(X, X) + \sigma_n^2 I]^{-1} k(X, \boldsymbol{x}^*))\end{aligned} \tag{2.41}$$

, where $k(X, \boldsymbol{x}^*) \in \mathbb{R}^N$ is a column vector with $k(X, \boldsymbol{x}^*)_i = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^*)$ and $k(X, X) \coloneqq \mathbf{K} \in \mathbb{R}^{N\times N}$ is the Gram matrix. Hence the posterior of $\boldsymbol{f}$ is also a GP, with $\boldsymbol{f}|X, \boldsymbol{y} \sim \mathcal{GP}(m_p, k_p)$ where

$$m_p(\boldsymbol{x}) = k(X, \boldsymbol{x})^\top [k(X, X) + \sigma_n^2 I]^{-1} \boldsymbol{y} \tag{2.42}$$

$$k_p(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - k(X, \boldsymbol{x})^\top [k(X, X) + \sigma_n^2 I]^{-1} k(X, \boldsymbol{x}')). \tag{2.43}$$

The posterior predictive distribution simply has an extra σ_n^2 term for the variance:

$$\begin{aligned}y^*|\boldsymbol{x}^*, X, \boldsymbol{y} \sim \mathcal{N}(&k(X, \boldsymbol{x}^*)^\top [k(X, X) + \sigma_n^2 I]^{-1} \boldsymbol{y}, \\ &k(\boldsymbol{x}^*, \boldsymbol{x}^*) - k(X, \boldsymbol{x}^*)^\top [k(X, X) + \sigma_n^2 I]^{-1} k(X, \boldsymbol{x}^*) + \sigma_n^2).\end{aligned} \tag{2.44}$$

The posterior mean is a *linear predictor*, since it is linear in $\boldsymbol{y}$ and also in $k(X, \boldsymbol{x}^*)$ i.e. it is of the form $m_p(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}^{(i)}, \boldsymbol{x})$. Hence for the SE kernel (c.f. Equation 2.2), the posterior mean can be thought of as a linear combination of Gaussians at each data point. The posterior variance is the difference between the prior variance and a positive definite term, which can be interpreted as how much the data X has explained (hence led to a decrease in variance). Although we would only use the posterior mean for the sake of prediction, we may obtain confidence intervals of the prediction using the predictive variance, giving us a measure of uncertainty. In the case of predictions at multiple test inputs, the predictive variance allows us to obtain a measure of covariance.

The hyperparameters $\boldsymbol{\eta}$ of the GP (the kernel parameters and the observation noise σ_n) can be optimised by Empirical Bayes (c.f. Section 2.2.3) i.e. by maximising the marginal likelihood of the data while integrating out the Gaussian Process. The marginal likelihood
$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|\boldsymbol{f}, X)p(\boldsymbol{f}(X)|X)d\boldsymbol{f}$ can be computed analytically by noting
$\boldsymbol{y}|X \sim \mathcal{N}(0, \mathbf{K} + \sigma_n^2 I)$ by conjugacy. Hence

$$\log p(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^\top(\mathbf{K} + \sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log\det(\mathbf{K} + \sigma_n^2 I) - \frac{N}{2}\log(2\pi). \tag{2.45}$$

Alternatively, we can directly optimise a metric of interest, that is a function of the posterior predictive distribution. Suppose we have a loss function $\mathcal{L}(\boldsymbol{y}_{test}, \boldsymbol{y}^*)$. Then we can minimise the expected loss with respect to the posterior predictive to tune the hyperparameters of the GP

$$\hat{\boldsymbol{\eta}} = \arg\min_{\boldsymbol{\eta}} \int \mathcal{L}(\boldsymbol{y}_{test}, y^*)p_{\boldsymbol{\eta}}(y^*|\boldsymbol{x}^*, X, y)dy^*. \tag{2.46}$$

For example if we wish to use the GP for classification, we may use the classification error as the loss function. However the advantage of using the marginal likelihood (or an approximation) is that we do not need to know the metric(s) of interest a priori, and it can also be useful when there are many metrics that we would like to optimise but unsure of how to weight them.

Hitherto, we have used a zero mean function for simplicity. When using a non-zero mean function for the prior GP, the predictive mean becomes

$$m_p(\boldsymbol{x}) = m(\boldsymbol{x}) + k(X, \boldsymbol{x})^\top[k(X, X) + \sigma_n^2 I]^{-1}(\boldsymbol{y} - m(\boldsymbol{x})) \tag{2.47}$$

and the predictive variance is unchanged from the zero-mean setting.

Scaling up GPs

GPs are appealing candidates for regression models, due to their *non-parametric* formulation: the complexity of the model (the dimensionality of the latent $\boldsymbol{f}(X)$) increases with the size of the dataset. However, the main disadvantage of GPs lies in their lack of scalability. The matrix inverses in the posterior predictive and the determinants of the Gram matrix in the marginal likelihood both rely on Cholesky decompositions of $\mathbf{K} + \sigma_n^2 I$, that costs $O(n^3)$ computational complexity and $O(n^2)$ memory. Hence there have been works in multiple directions that scale up GPs by allowing lower time and memory complexity.

One notable approach to scaling up GPs can be categorised as *inducing point methods* [Seeger et al., 2003, Lázaro-Gredilla et al., 2010, Titsias, 2009, Quinonero-Candela and Rasmussen, 2005]. These approaches rely on a selection of $m \ll N$ inducing points in the input space $\mathcal{X}$ that attempt to explain all the covariance in the Gram matrix of the kernel; the kernel is evaluated for each pair of inducing points and also between the inducing points and the data, giving matrices $\mathbf{K}_{mm}, \mathbf{K}_{mN} = \mathbf{K}_{Nm}^\top$. Most such methods rely on the *Nyström approximation* $\hat{\mathbf{K}} \coloneqq \mathbf{K}_{Nm}(\mathbf{K}_{mm})^\dagger \mathbf{K}_{mN}$ of the Gram matrix [Williams and Seeger, 2001, Drineas and Mahoney, 2005], where $\dagger$ denotes the pseudo-inverse. Note that this approximation admits a low-rank form $\boldsymbol{\Phi}^\top \boldsymbol{\Phi}$ (using the Cholesky decomposition of $\mathbf{K}_{mm}$), and this allows efficient computation of determinants and inverses in $O(Nm^2)$ time and memory via the following matrix identities:

Lemma 1 (Woodbury's Matrix Inversion Lemma)**.** *[Woodbury, 1950]*

$$(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \sigma_n^2 I)^{-1} = \sigma_n^{-2} I - \sigma_n^{-2} \boldsymbol{\Phi}^\top (\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma_n^2 I)^{-1} \boldsymbol{\Phi}$$

Lemma 2 (Sylvester's Determinant Theorem)**.** *[Sylvester, 1851]*

$$\det(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \sigma_n^2 I) = (\sigma^2)^{n-m} \det(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma_n^2 I)$$

Another line of scaling up GPs is to use an unbiased stochastic low-rank approximation of the Gram matrix, namely *random Fourier features* (RFF) Rahimi and Recht [2007]. We use the following theorem to construct these features:

Theorem 1 (Bochner's Theorem)**.** *(e.g. [Edwards, 1979]) A stationary kernel k(d) is positive definite if and only if k(d) is the Fourier transform of a non-negative measure.*

For RFF the kernel can be approximated by the inner product of random features given by samples from its spectral density, in a Monte Carlo approximation, as follows:

$$\begin{aligned} k(\boldsymbol{x}-\boldsymbol{x}') = \int_{\mathbb{R}^D} e^{i\boldsymbol{v}^\top(\boldsymbol{x}-\boldsymbol{x}')} d\mathbb{P}(v) \propto \int_{\mathbb{R}^D} p(\boldsymbol{v}) e^{i\boldsymbol{v}^\top(\boldsymbol{x}-\boldsymbol{x}')} dv &= \mathbb{E}_{p(v)}[e^{i\boldsymbol{v}^\top\boldsymbol{x}}(e^{i\boldsymbol{v}^\top\boldsymbol{x}'})^*] \\ &= \mathbb{E}_{p(v)}[Re(e^{i\boldsymbol{v}^\top\boldsymbol{x}}(e^{i\boldsymbol{v}^\top\boldsymbol{x}'})^*)] \\ &\approx \frac{1}{n}\sum_{k=1}^{n} Re(e^{i\boldsymbol{v}^{(k)\top}\boldsymbol{x}}(e^{i\boldsymbol{v}^{(k)\top}\boldsymbol{x}'})^*) \\ &= \mathbb{E}_b[\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')^*], \end{aligned}$$

where different choices of $\boldsymbol{\phi}$ are possible. A common choice is $\boldsymbol{\phi}(\boldsymbol{x}) = \sqrt{\frac{2}{n}}(\cos(\boldsymbol{v}^{(1)\top}\boldsymbol{x} + b_1), \ldots, \cos(\boldsymbol{v}^{(m)\top}\boldsymbol{x} + b_m))$ with spectral frequencies $\boldsymbol{v}^{(k)}$ being *iid* samples from $p(v)$ and b_k *iid* samples from $U[0, 2\pi]$, also used in Chapters 3 and 4. Comparison with other choices of $\boldsymbol{\phi}$ can be found in Sutherland and Schneider [2015].
For a one dimensional squared exponential kernel $k(x, x') = \sigma_f^2 \exp\Big(-\frac{(x-x')^2}{2l^2}\Big)$, the spectral density $p(v)$ is $\mathcal{N}(0, l^{-2})$. Hence we use features
$\boldsymbol{\phi}(x) = \sigma_f\sqrt{\frac{2}{n}}(\cos(v^{(1)}x + b_1), \ldots, \cos(v^{(m)}x + b_m))$ where $v^{(k)}$ are *iid* samples from $\mathcal{N}(0, l^{-2})$ and b_k are *iid* samples from $U[0, 2\pi]$. For RFF the above matrix identities can also be leveraged to reduce time and memory complexity to $O(Nm^2)$.

A qualitatively different approach to scaling up GPs is to compute exact matrix inverses and determinants exploiting special structure in the inputs and the kernel. When the inputs lie on a D-dimensional Cartesian grid (that need not be equi-spaced), it can be shown that exact inference can be performed in $O(N^{(1+1/D)}D + N^{3/D})$ time and $O(N^{(1+1/D)}D + N^{2/D})$ memory for product kernels using Kronecker product identities [Saatçi, 2011]. These are kernels that can be expressed in terms of products across the input dimensions, and most common kernels (e.g. squared exponential kernel) are of this type. However note that there is no gain for $D = 1$, and the number of grid points (and hence the number of data points) scales exponentially in D, so applicability is limited for higher dimensions.

These aforementioned approaches for scaling up GPs are used throughout in Chapters 3 and 4.

Conjugate Gradients for optimisation in GPs

In GPs, one set of parameters that have to be learned from the data are the hyper-parameters, including the kernel hyperparameters and the observation noise. These are learned by optimising the given loss (e.g. log marginal likelihood), and an obvious

choice of optimisation algorithm is *gradient descent*: given hyperparameters $\boldsymbol{\eta} \in \mathbb{R}^p$ and a loss $\mathcal{L}(\boldsymbol{\eta})$, we can choose an initial estimate $\boldsymbol{\eta}_{(0)}$ and take iterative updates $\boldsymbol{\eta}_{(i+1)} = \boldsymbol{\eta}_{(i)} + \alpha_{(i)}\mathcal{L}'(\boldsymbol{\eta}_{(i)})$ where the step size $\alpha_{(i)}$ can either be fixed or determined at every step using *line search*, so that $\mathcal{L}(\boldsymbol{\eta}_{(i+1)})$ is minimal. Every step of gradient descent has time complexity $O(p)$, and requires few evaluations of the loss, hence it is usually chosen in the large p scenario. However in the context of GPs, p is usually small (a GP typically only has a handful of kernel hyperparameters), hence other optimisation algorithms may be more suitable.

A popular choice for GP hyperparameter optimisation is the *non-linear conjugate gradient method* [Fletcher and Reeves, 1964], that essentially assumes that the loss function is locally quadratic. To introduce this method, we first overview the *conjugate gradient method* [Hestenes and Stiefel, 1952] that is used for fast optimisation of quadratic expressions. In fact this can also be used for solving linear systems involving Gram matrices, as we will show below. Most of what follows is a summary of Shewchuk et al. [1994].

The problem at hand for the conjugate gradient (CG) method is to optimise a *quadratic form* $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top A\boldsymbol{x} - \boldsymbol{b}^\top\boldsymbol{x} + c \in \mathbb{R}$. $f(\boldsymbol{x})$ is a paraboloid (shaped like a bowl) if and only if A is a positive definite matrix, i.e. $\boldsymbol{x}^\top A\boldsymbol{x} > 0 \ \forall \boldsymbol{x} \neq 0$. Note that b, c determine the location of the minimum, but only A affects the shape of the paraboloid. By taking derivatives with respect to $\boldsymbol{x}$, it is easy to see that the optimal $\boldsymbol{x}^* = A^{-1}b$, hence the relevance of CG for computing expressions such as $K^{-1}v$ where K is the Gram matrix. This observation is used in Chapter 4. Note that matrix inversion takes $O(p^3)$ computation, hence is can be inefficient for large p. A faster alternative that CG employs is to take iterative updates to an initial guess $\boldsymbol{x}_{(0)}$. First let us introduce some notation

$$
\begin{aligned}
\boldsymbol{e}_{(i)} =& \boldsymbol{x}_{(i)} - \boldsymbol{x}^* \text{ (error: how far we are from the true solution)} \\
\boldsymbol{r}_{(i)} =& b - A\boldsymbol{x}_{(i)} \text{ (residual)} \\
=& - A\boldsymbol{e}_{(i)} \text{ (error transformed to same space as } b) \\
=& - f'(\boldsymbol{x}_{(i)}) \text{ (direction of steepest descent)}
\end{aligned}
$$

In steepest descent, we have updates $\boldsymbol{x}_{(i+1)} = \boldsymbol{x}_{(i)} + \alpha_{(i)}\boldsymbol{r}_{(i)}$ where $\alpha_{(i)}$ should be chosen such that $f(\boldsymbol{x}_{(i+1)})$ is minimal. This is precisely the point at which $\boldsymbol{r}_{(i)}$ and $f'(\boldsymbol{x}_{(i+1)})$ are orthogonal. It is straightforward to see that $\alpha_{(i)} = \frac{\boldsymbol{r}_{(i)}^\top \boldsymbol{r}_{(i)}}{\boldsymbol{r}_{(i)}^\top A\boldsymbol{r}_{(i)}}$. However steepest descent often takes steps in similar directions as earlier steps. This is wasteful, hence the idea of conjugate gradients is to choose a set of linearly independent search

directions $\boldsymbol{d}_{(0)}, \ldots, \boldsymbol{d}_{(p-1)}$ and take exactly one step of the right length in each step direction. In particular, we would like to make the search directions *conjugate*, or A-orthogonal, such that $\boldsymbol{d}_{(i)}^\top A \boldsymbol{d}_{(j)} = 0$ for $i \neq j$ (two conjugate vectors become orthogonal when the paraboloid is stretched such that the contour lines become circular). It is easy to show that these directions are linearly independent if A is positive definite.

Given such pairwise conjugate search directions, we would like $\boldsymbol{e}_{(i+1)}$ conjugate to $\boldsymbol{d}_{(i)}$, equivalent to finding a minimum of f along $\boldsymbol{d}_{(i)}$ from $\boldsymbol{x}_{(i)}$. It is again easy to show that the right step size $\alpha_{(i)} = \frac{\boldsymbol{d}_{(i)}^\top \boldsymbol{r}_{(i)}}{\boldsymbol{d}_{(i)}^\top A \boldsymbol{d}_{(i)}}$, and can be proven that this procedure reaches $\boldsymbol{x}^*$ in at most p steps. This holds because the ith update is equivalent to removing the $\boldsymbol{d}_{(i)}$ component when $\boldsymbol{e}_{(i)}$ is written as a linear combination of the search directions; say $\boldsymbol{e}_{(i)} = \sum_j \delta_j \boldsymbol{d}_{(j)}$ (by linear independence of directions), then

$$\begin{aligned}
\boldsymbol{d}_{(i)}^\top A \boldsymbol{e}_{(i)} =& \sum_j \delta_j \boldsymbol{d}_{(i)}^\top A \boldsymbol{d}_{(j)} \\
=& \delta_i \boldsymbol{d}_{(i)}^\top A \boldsymbol{d}_{(i)} \text{ by conjugacy of search directions} \\
\Rightarrow \delta_i =& \frac{\boldsymbol{d}_{(i)}^\top A \boldsymbol{e}_{(i)}}{\boldsymbol{d}_{(i)}^\top A \boldsymbol{d}_{(i)}} = -\alpha_{(i)}.
\end{aligned}$$

Hence it follows that the error $\boldsymbol{e}_{(i+1)}$ is conjugate to all past search directions $\boldsymbol{d}_{(j)}$ for $j \leq i$, and equivalently $\boldsymbol{r}_{(i+1)}$ are orthogonal to $\boldsymbol{d}_{(j)}$ for $j \leq i$. The set of pairwise conjugate search directions can be constructed iteratively via the *conjugate Gram-Schmidt process* (CGSP)

1. Choose p linearly independent vectors $\boldsymbol{u}_{(0)}, \ldots, \boldsymbol{u}_{(p-1)}$.
2. Set $\boldsymbol{d}_{(0)} = \boldsymbol{u}_{(0)}$ and compute $\boldsymbol{d}_{(i)} = u_{(i)} + \sum_{k=0}^{i-1} \beta_{ik} \boldsymbol{d}_{(k)}$ recursively where $\beta_{ij} = -\frac{\boldsymbol{u}_{(i)}^\top A \boldsymbol{d}_{(j)}}{\boldsymbol{d}_{(i)}^\top A \boldsymbol{d}_{(j)}}$ (i.e. take $\boldsymbol{u}_{(i)}$ and subtract out all components that are not conjugate to previous search directions).

The problem with CGSP is that a total of $O(p^3)$ operations are required to generate the full set, and old search directions must be kept in memory to construct each new one (so $O(p^2)$ memory). The *conjugate gradient method* (CG) avoids this issue by using the residuals to construct the $\boldsymbol{d}_{(i)}$ i.e. setting $\boldsymbol{u}_{(i)} = \boldsymbol{r}_{(i)}$. Using the recursive formula for $\boldsymbol{d}_{(i)}$ in CGSP and that $\boldsymbol{r}_{(i+1)}^\top \boldsymbol{d}_{(j)} = 0$ for $j \leq i$, we can show that the $\boldsymbol{r}_{(i)}$ are mutually orthogonal and that $\boldsymbol{r}_{(i+1)}$ is conjugate to all previous search directions except $\boldsymbol{d}_{(i)}$. Hence $\beta_{i+1,j} = 0 \forall j < i$. In summary, CG is

1. $\boldsymbol{d}_{(0)} = \boldsymbol{r}_{(0)} = \boldsymbol{b} - A\boldsymbol{x}_{(0)}$.

2. For $i = 0, \ldots, p-1$:

   (a) $\boldsymbol{x}_{(i+1)} = \boldsymbol{x}_{(i)} + \alpha_{(i)}\boldsymbol{d}_{(i)}$ where $\alpha_{(i)} = \frac{\boldsymbol{d}_{(i)}^\top \boldsymbol{r}_{(i)}}{\boldsymbol{d}_{(i)}^\top A \boldsymbol{d}_{(i)}}$.

   (b) $\boldsymbol{r}_{(i+1)} = \boldsymbol{r}_{(i)} - \alpha_{(i)} A\boldsymbol{d}_{(i)}$, $\beta_{(i+1)} = \frac{\boldsymbol{r}_{(i+1)}^\top \boldsymbol{r}_{(i+1)}}{\boldsymbol{r}_{(i)}^\top \boldsymbol{r}_{(i)}}$, $\boldsymbol{d}_{(i+1)} = \boldsymbol{r}_{(i+1)} + \beta_{(i+1)}\boldsymbol{d}_{(i)}$.

Here the time and space complexity per iteration has been reduced from $O(p^2)$ to $O(m)$ where m are the number of non-zeros of A. For GPs, usually $A = K$ or $A = K + \sigma^2 I$, hence $m = p^2$. Note that $\boldsymbol{r}_{(i)}$ is not being calculated from $\boldsymbol{b} - A\boldsymbol{x}_{(i)}$, hence floating point error may accumulate in practice. To resolve this, it is useful to update $\boldsymbol{r}_{(i)}$ with $\boldsymbol{b} - A\boldsymbol{x}_{(i)}$ occasionally.

CG is commonly used for large problems where it is infeasible to run it for p iterations. Hence we would like to know the rate of convergence of CG. Shewchuk et al. [1994] shows that the maximum number of iterations to reach error ϵ is $\lceil \frac{1}{2}\sqrt{\mathcal{K}(A)} \log \frac{2}{\epsilon}) \rceil$ where $\mathcal{K}(A)$ is the condition number of A, the ratio of the maximal and minimal singular values of A. Hence CG has time complexity $O(m\sqrt{\mathcal{K}})$ and space complexity $O(m)$.

A common trick used for CG is *preconditioning*, where given some symmetric positive definite matrix M that is easy to invert, we solve $M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}$ instead of $A\boldsymbol{x} = \boldsymbol{b}$. The reason for doing so is that CG on this equivalent problem can converge more quickly than on the original problem if $\mathcal{K}(M^{-1}A) \ll \mathcal{K}(A)$. However $M^{-1}A$ need not be symmetric nor positive definite, hence CG might not be applicable. The workaround is to use $M = EE^\top$ where E is a positive definite square matrix e.g. the Cholesky decomposition (upper triangular and hence positive definite), then transform $A\boldsymbol{x} = \boldsymbol{b}$ into $E^{-1}AE^{-\top}\hat{\boldsymbol{x}} = E^{-1}\boldsymbol{b}$ where $\hat{x} = E^\top \boldsymbol{x}$. Noting that $E^{-1}AE^{-\top}$ is positive definite, we can solve for $\hat{\boldsymbol{x}}$ using CG, then solve quickly for $\boldsymbol{x}$ by Gaussian elimination. It can also be shown that $\mathcal{K}(M^{-1}A) = \mathcal{K}(E^{-1}AE^{-\top})$, so the time complexity is maintained.

Having developed the method of conjugate gradients, it can be used to minimise any continuous function f for which the gradient f' can be obtained, by taking its quadratic approximation at the local neighbourhood of an initial estimate and performing CG updates. Thus we introduce *non-linear conjugate gradients*, for which there are three main changes from CG: the recursive formula for $\boldsymbol{r}_{(i)}$ cannot be used, α is more difficult to compute, and there are several possible choices for $\beta_{(i)}$. Consequently we set $\boldsymbol{r}_{(i)} = -f'(\boldsymbol{x}_{(i)})$, with $\boldsymbol{d}_{(i)}$ computed as in CG, and set $\alpha_{(i)}$ to be the value that approximately minimises $f(\boldsymbol{x}_{(i)} + \alpha_{(i)}\boldsymbol{d}_{(i)})$. While other choices are possible, a standard method is to use the Newton-Raphson method to solve $f'(\boldsymbol{x}_{(i)} + \alpha\boldsymbol{d}_{(i)}) = 0$

for α. The Hessian that is involved can be approximated by finite differences. The two choices for β are

1. Fletcher-Reeves: $\beta_{(i+1)}^{\mathrm{FR}} = \frac{\boldsymbol{r}_{(i+1)}^\top \boldsymbol{r}_{(i+1)}}{\boldsymbol{r}_{(i)}^\top \boldsymbol{r}_{(i)}}$.
2. Polak-Ribiére $\beta_{(i+1)}^{\mathrm{PR}} = \frac{\boldsymbol{r}_{(i+1)}^\top (\boldsymbol{r}_{(i+1)} - \boldsymbol{r}_{(i)})}{\boldsymbol{r}_{(i)}^\top \boldsymbol{r}_{(i)}}$.

Note that for linear CG, these two are equivalent since $\boldsymbol{r}_{(i+1)}^\top \boldsymbol{r}_{(i)} = 0$, and that the Hessian just amounts to the matrix A. Usually FR converges if the starting point is sufficiently close to the mean, whereas PR can, in rare cases, cycle infinitely. However PR often converges more quickly [Shewchuk et al., 1994]. In summary, non-linear CG works as follows

1. $\boldsymbol{d}_{(0)} = \boldsymbol{r}_{(0)} = -f'(\boldsymbol{x}_{(0)})$.
2. For $i = 0, \ldots, p-1$:
   (a) $\boldsymbol{x}_{(i+1)} = \boldsymbol{x}_{(i)} + \alpha_{(i)} \boldsymbol{d}_{(i)}$ where $\alpha_{(i)}$ is chosen to minimise $f(\boldsymbol{x}_{(i)} + \alpha_{(i)} \boldsymbol{d}_{(i)})$ (via e.g. Newton-Raphson method).
   (b) $\boldsymbol{r}_{(i+1)} = -f'(\boldsymbol{x}_{(i+1)})$, $\beta_{(i+1)} = \beta_{(i+1)}^{\mathrm{FR}}$ or $\beta_{(i+1)}^{\mathrm{PR}} \wedge 0$, $\boldsymbol{d}_{(i+1)} = \boldsymbol{r}_{(i+1)} + \beta_{(i+1)} \boldsymbol{d}_{(i)}$.

Since now convergence is not guaranteed by p iterations, it makes sense to restart CG every p or fewer iterations, especially if p is small.

2.2.10 Probabilistic Matrix Factorisation for Collaborative Filtering

We cover probabilistic matrix factorisation models for collaborative filtering [Salakhutdinov and Mnih, 2008a,b] as further examples of probabilistic models in Bayesian machine learning.

Collaborative filtering (CF) defines a branch of techniques for tackling the following supervised learning problem: making predictions (filtering) about the preferences of a user, based on information regarding the preferences of many users (collaboration). We are given data in the form of a partially observed rating matrix R, where R_{ij} is the rating of user u_i on movie v_j for $i = 1, \ldots, N, j = 1, \ldots, M$. CF aims to predict missing entries of R by only using the observed entries.

One of the most successful approaches to CF is the idea of applying matrix factorisation to the rating matrix [Billsus and Pazzani, 1998, Koren, 2009, Piotte and Chabbert, 2009, Töscher et al., 2009]. The idea is to factorise the $N \times M$ rating matrix R as a product of two matrices U and $V^\top$, where $U \in \mathbb{R}^{N \times D}$ and $V \in \mathbb{R}^{M \times D}$ with D being a hyperparameter to be tuned. The interpretation of U and V is that each row are user and movie features respectively, and their inner product models the rating of the particular user on the particular movie. Hence we learn U and V so that the error on the observed ratings is minimised. Then we simply fill in the blanks of R by multiplying out U and $V^\top$.

Probabilistic Matrix Factorisation (PMF) [Salakhutdinov and Mnih, 2008a] offers a probabilistic model that formulates U and V as latent variables with Gaussian priors, and uses MAP estimates of U and V to make predictions. Denoting row i of U as U_i and row j of V as V_j, the likelihood of the ratings is modelled as a Gaussian with the mean equal to the inner product of U_i and V_j, with fixed variance

$$p(R_{ij}|U_i, V_j) = \mathcal{N}(U_i V_j^\top, \sigma^2). \tag{2.48}$$

Note that although the ratings are usually integer-valued, ranging between 1 and 5, it is modelled by a continuous random variable. This is, however, justified as there is no requirement that the prediction must also be integer-valued. From a recommender system application point of view, is in fact beneficial to model R_{ij} as a continuous random variable, since then the non-integer part of the prediction would be helpful in distinguishing the top recommendations from the decent ones. Also for simplicity, the model assumes conditional independence across the ratings R_{ij} given U and V. Hence

$$p(R|U, V) = \prod_{(ij)} \mathcal{N}(R_{ij}|U_i V_j^\top, \sigma^2), \tag{2.49}$$

where the product ranges over pairs of user/movie indices in the training set.

The model places Gaussian priors with zero mean and spherical covariance on both U and V

$$p(U) = \prod_{i=1}^{N} \mathcal{N}(U_i|0, \sigma_U^2 I) \qquad p(V) = \prod_{j=1}^{M} \mathcal{N}(V_j|0, \sigma_V^2 I). \tag{2.50}$$

These priors can be interpreted as a natural regularisation induced by our Bayesian model, which can help reduce overfitting. We will see below how the values of σ_U, σ_V can be interpreted as the degree of regularisation.

By Bayes' Theorem (Equation 2.4), the log posterior of U and V can be calculated up to the normalising constant:

$$\log p(U, V|R) = -\frac{1}{2\sigma^2}\sum_{(ij)}(R_{ij} - U_i V_j^\top)^2 - \frac{1}{2\sigma_U^2}\sum_{i=1}^{N}\|U_i\|_2^2 - \frac{1}{2\sigma_V^2}\sum_{j=1}^{M}\|V_j\|_2^2 + C, \quad (2.51)$$

where C is the log normalising constant that is independent of U, V. Maximising the log-posterior over U and V (MAP) is then equivalent to minimising the sum of squared errors with quadratic regularisation terms

$$E = \frac{1}{2}\sum_{(ij)}(R_{ij} - U_i V_j^\top)^2 + \frac{\lambda_U^2}{2}\sum_{i=1}^{N}\|U_i\|_2^2 + \frac{\lambda_V^2}{2}\sum_{j=1}^{M}\|V_j\|_2^2, \quad (2.52)$$

where $\lambda_U = \sigma^2/\sigma_U^2, \lambda_V = \sigma^2/\sigma_V^2$. Note that this reduces the number of free hyperparameters from three to two. Moreover, note that using Gaussian priors on U and V combined with MAP inference is equivalent to L2 regularisation (i.e. penalising the L2 norm of the parameters), which encourages the entries of U and V to be small. Intuitively, this discourages extreme predictions for unobserved entries, hence acts as a sensible regularisation term. The MAP estimates of U and V are computed via gradient descent, and early stopping is used to control for overfitting i.e. training is halted when the error on the validation set reaches its minimum.

One difficulty of learning the PMF model is that the hyperparameters λ_U, λ_V and D must be chosen manually to values that minimise the validation error, and the choice for these hyperparameters are crucial for good generalisation to test data. Another potential problem of the PMF algorithm is that we use MAP estimates for U and V, a method that is prone to overfitting (despite the L2 regularisation) as it finds a point estimate of U and V and does not take into account the uncertainty in U and V.

Bayesian Probabilistic Matrix Factorisation (BayesPMF) [Salakhutdinov and Mnih, 2008b] addresses these concerns by extending the PMF model to a hierarchical Bayesian model. We build on the same likelihood as PMF

$$p(R|U, V) = \prod_{(ij)}\mathcal{N}(R_{ij}|U_i V_j^T, \alpha^{-1}). \quad (2.53)$$

To avoid having to set the regularisation parameters λ_U and λ_V, manually, it would be sensible to create a further level of parameterisation, by placing priors on the distributional parameters of U and V. In the BayesPMF model, the priors on U and

V are now non-centered Gaussians, with further priors on both the mean and the precision, the inverse of the covariance matrix

$$p(U) = \prod_{i=1}^{N} \mathcal{N}(U_i|\mu_U, \Lambda_U^{-1}) \qquad p(V) = \prod_{j=1}^{M} \mathcal{N}(V_j|\mu_V, \Lambda_V^{-1}) \tag{2.54}$$

$$\begin{aligned} p(\Theta_U|\Theta_0) &= p(\mu_U|\Lambda_U)p(\Lambda_U) \\ &= \mathcal{N}(\mu_U|\mu_0, (\beta_0\Lambda_U)^{-1})\mathcal{W}(\Lambda_U|W_0, \nu_0) \end{aligned} \tag{2.55}$$

$$\begin{aligned} p(\Theta_V|\Theta_0) &= p(\mu_V|\Lambda_V)p(\Lambda_V) \\ &= \mathcal{N}(\mu_V|\mu_0, (\beta_0\Lambda_V)^{-1})\mathcal{W}(\Lambda_U|W_0, \nu_0). \end{aligned} \tag{2.56}$$

Note that we have placed Gaussian-Wishart (G-W) priors on $\Theta_U = \{\mu_U, \Lambda_U\}$ and $\Theta_V = \{\mu_V, \Lambda_V\}$. This is because the G-W distribution is the conjugate prior to the Gaussian likelihood with unknown mean and unknown precision. In other words, this choice of prior ensures that the distribution of Θ_U given U, V and Θ_V given U, V are both G-W, allowing us to sample from these conditional distributions without resorting to approximate methods. The set of hyperparameters (that must be fixed) are defined to be $\Theta_0 = \{\mu_0, \nu_0, W_0, \beta_0\}$. Although one may think that in fact we have ended up with a larger set of hyperparameters to tune than PMF, it is usually the case that the performance of models are less sensitive to hyperparameters for hierarchical Bayesian models than 'flatter' models, hence tuning requires less effort in BayesPMF.

Now the predictive distribution of a new rating R_{ij}^* is obtained by integrating out the latent variables U, V, Θ_U, Θ_V from the joint

$$p(R_{ij}^*|R, \Theta_0) = \mathbb{E}_{p(\Theta_U, \Theta_V|\Theta_0)}\mathbb{E}_{p(U,V|R,\Theta_U,\Theta_V)}[p(R_{ij}^*|U_i, V_j)]. \tag{2.57}$$

Exact evaluation of this distribution is analytically intractable, hence we must rely on approximate inference using a Monte Carlo approximation

$$p(R_{ij}^*|R, \Theta_0) \approx \frac{1}{T}\sum_{t=1}^{T} p(R_{ij}^*|U_i^{(t)}, V_j^{(t)}). \tag{2.58}$$

where T is the number of samples from the (approximate) posterior. Here each sample $\{U_i^{(t)}, V_j^{(t)}\}$ is generated by running a Markov chain with stationary distribution equal to the posterior of $\{U, V, \Theta_U, \Theta_V\}$. Gibbs sampling is used (c.f. Section 2.2.6) for each sample $\{U_i^{(t)}, V_j^{(t)}\}$. MCMC methods are rarely used for large-scale learning as they

are deemed too computationally demanding. Nonetheless in the BayesPMF model, the use of G-W priors, which gives closed form conditionals and thus allows fast sampling, gives rise to a computationally feasible implementation of Gibbs sampling. Details and derivations of the conditional distributions that give the transition kernels for the Gibbs sampler are given in Salakhutdinov and Mnih [2008b].

2.3 Interpretable Machine Learning

2.3.1 What is Interpretability? Rather, what properties are Interpretable?

As Machine Learning (ML) solutions to real-world problems are becoming increasingly common, ML researchers are striving to better understand the models that they develop. The community has been using the term *interpretability* to describe models and methods that help us achieve this rather vague goal, but there is no consensus for a definition of this term. Doshi-Velez [2017] define interpretability as "the ability to explain or to present in understandable terms to a human". This could refer to explanations of the predictions given by the system, or of the internal workings of the model, or both. Lipton [2016], on the other hand, claims that the motives and descriptions of interpretability is diverse and discordant, hence it is more insightful to think of interpretability as several distinct but related ideas instead of a single definition. In particular the work presents a thorough but non-exhaustive list of desiderata that we seek from interpretable models. We present the list below and provide examples of works that display such properties

1. *Trust*: confidence that a model will perform well with respect to real objectives and scenarios, and that it gives accurate predictions when humans can also be accurate [Kim et al., 2015, Ribeiro et al., 2016]

2. *Transferability*: robustness to adversarial examples and ability to transfer learned skills to unfamiliar situations [Goodfellow et al., 2014b, Madry et al., 2017]

3. *Fair and ethical decision making*: decisions made by algorithms conform to ethical standards [Zemel et al., 2013, Louizos et al., 2015]

4. *Causality*: help form causal hypotheses that can be tested [Rani et al., 2006]

5. *Informativeness*: provide useful information for human decision-making [Huysmans et al., 2011]

Properties 1-3 are closely related to the notion of *safety* when it comes to ML systems [Amodei et al., 2016]. Based on these requirements, Lipton [2016] lists properties that can be called *interpretable*, and categorises them under the headings *transparency* and *post-hoc interpretability*:

1. *Transparency*: understanding the mechanism of the model/algorithm
   (a) *simulatability*: easily understandable computation
   (b) *decomposability*: each part of model (input/parameters/computation) admits an intuitive explanation
   (c) *algorithmic transparency*: can guarantee that the algorithm will work on unseen data points/datasets
2. *Post-hoc interpretability*: can give useful information about the model's mechanism and/or its predictions
   (a) text explanation
   (b) visualisation: qualitative understanding of model
   (c) local (per-data point) explanation
   (d) explanation by example e.g. finding points which the model views to be similar

2.3.2 Interpretable Models in Probabilistic Machine Learning

In the remaining chapters of the thesis, we will describe models in probabilistic machine learning that show the aforementioned properties of interpretability.

The Tucker GP introduced in Chapter 3 satisfies decomposability, in that the regression function is additive, with the additive components arising from a product of features of each covariate. Thus the model allows us to analyse each additive component and investigate the multiplicative interactions between the covariates within the component. There is also an element of post-hoc interpretability, since we may visualise the different additive components to obtain a qualitative understanding of the role of each component for the prediction.

The SSDGP algorithm introduced in Chapter 4 is an algorithm for automated exploratory data analysis via automated model selection. The GP models chosen by the algorithm show post-hoc interpretability in the form of text explanations, as the model can be translated to natural language that describes the patterns in the data learned by the model. This again arises from the decomposability of the GP's kernel function, whereby 1) a GP with an additive kernel function can be written as a sum of GPs with these respective kernels and 2) when the kernel function is multiplied by one of the base kernels, the model's predictions are modified in a consistent manner (details given in Chapter 4). Hence each additive component can be analysed separately, and the multiplicative components can be described in natural language via hard-coded rules.

Chapter 5 describes FactorVAE, a model for unsupervised representation learning, namely disentangling factors of variation in image data. The model shows decomposability in that the different dimensions of the learned representation quantifies the value of different factors of variation present in the data. For the image datasets in the paper, these factors are properties such as size, orientation, and position of the object inside the image. The model also allows for post-hoc visualisation of each latent dimension; we can traverse each latent dimension and pass the traversals through the VAE decoder to visualise the effect of changing the latents in the pixel space. The learned representations also give a sense of which images the model deems to be semantically similar, by looking at Euclidean distance in the latent space as opposed to the pixel space.

The Attentive Neural Process (ANP) introduced in Chapter 6 incorporates attention into Neural Processes [Garnelo et al., 2018b], allowing the decoder to focus on context points that are relevant for a given target prediction. This mechanism of attention admits such an intuitive explanation, demonstrating decomposability of the ANP. The modular nature of the model that is composed of the encoder that summarises the context, and the decoder that uses the summary to make predictions also displays the simulatable nature of the model. Moreover the model shows decomposability in that the latent variable models the distribution over functions – each value of the latent corresponds to a function. The attention also allows for visualisations of which context points are relevant for a given prediction, providing a visual *and* local explanation. In the case of multi-head attention, the visualisation of attention maps per head can help understand the role of each head in making the prediction. The quantification of predictive uncertainty in the model also aids the algorithmic transparency of the

model, as the predictive variance can be seen as a measure of the model's confidence in the prediction.

Chapter 3

Collaborative Filtering with Side Information: a Gaussian Process Perspective

The following chapter is a self-contained paper:

> **Kim, H.**, Lu, X., Flaxman, S. and Teh, Y.W., 2016, May. Collaborative Filtering with Side Information: a Gaussian Process Perspective. arXiv preprint arXiv:1605.07025.

There is a statement of authorship at the end of this chapter.

Abstract

We tackle the problem of collaborative filtering (CF) with side information, through the lens of Gaussian Process (GP) regression. Driven by the idea of using the kernel to explicitly model user-item similarities, we formulate the GP in a way that allows the incorporation of low-rank matrix factorisation, arriving at our model, the *Tucker Gaussian Process* (TGP). Consequently, TGP generalises classical Bayesian matrix factorisation models, and goes beyond them to give a natural and elegant method for incorporating side information, giving enhanced predictive performance for CF problems. Moreover we show that it is a novel model for regression, especially well-suited to grid-structured data and problems where the dependence on covariates is close to being separable.

3.1 Introduction

Collaborative filtering (CF) defines a branch of techniques for tackling the following supervised learning problem: making predictions (filtering) about the preferences of a user, based on information regarding the preferences of many users (collaboration). We are given data in the form of a partially observed rating matrix R, where R_{ij} is the rating of user u_i on movie v_j for $i = 1, \ldots, n_1, j = 1, \ldots, n_2$. CF aims to predict missing entries of R by only using the observed entries. Hitherto, matrix factorisation approaches [Billsus and Pazzani, 1998, Koren, 2009, Piotte and Chabbert, 2009, Töscher et al., 2009] have been the basis for many successful CF models. These model R as a product of two low rank matrices $R \approx UV^\top$, hence $R_{ij} \approx \sum_k U_{ik}V_{jk}$. On the other hand, content-based filtering predicts user ratings based on attributes of users (e.g. age, sex) and items (e.g. genre). CF with side information is a combination of the two, aiming to predict user ratings using both ratings data and user/item attributes.

There has been a wide range of work on CF with side information, mostly building on the framework of matrix factorisation. Suppose user/item side information is given in the form of feature matrices $F = [\omega(u_1), ..., \omega(u_{n_1})]^\top \in \mathbb{R}^{n_1 \times r}$ and $G = [\omega'(v_1), ..., \omega'(v_{n_2})]^\top \in \mathbb{R}^{n_2 \times r}$ where u_i, v_j represent user and item identities. *Matrix co-factorization* Singh and Gordon [2008] attempt to factorise F, G and R simultaneously, whereas the *Regression-based Latent Factor Model* [Agarwal and Chen, 2009] assumes instead that U and V are linear in F and G. *Bayesian Matrix Factorization with Side Information* (BMFSI) [Porteous and Welling, 2010] gives an additive model in the sense that R is assumed to be the sum of the standard matrix factorisation prediction $UV^\top$ and linear contributions of F and G. *Hierarchical Bayesian Matrix Factorization with Side Information* [Park et al., 2013] is an extension of BFMSI with Gaussian-Wishart hyperpriors on the prior mean and variance of U and V.

Gaussian Processes (GPs) are a popular class of Bayesian nonparametric priors over functions [Rasmussen and Williams, 2005], and have served as flexible models across a range of machine learning tasks, e.g. regression, classification, dimensionality reduction [Lawrence, 2004] and CF [Lawrence and Urtasun, 2009, Yu et al., 2006]. In a regression setting with input and output pairs, a key advantage of GPs is that we can use the kernel to explicitly model similarity in the outputs between a pair of input values. We use this to model similarity between users/items given side information, forming the outset of the paper. We model the ratings as $R_{ij} \sim \mathcal{N}(f(u_i, v_j), \sigma^2)$ and

$f \sim \mathcal{GP}(k_1 \times k_2)$ where kernels k_1 and k_2 model user and item similarities respectively. However, a direct GP regression application, that optimises the marginal likelihood of observed ratings $p(\mathbf{R}|\boldsymbol{u}, \boldsymbol{v})$, is infeasible due to the $O(N^3)$ computational cost of evaluating gradients and $O(N^2)$ cost for producing each new prediction; for most CF problems, the number of ratings N ranges from a hundred thousand to hundreds of millions. Low-rank matrix factorisation remedies this problem, and also underlies the state of the art approaches for CF. Hence it is natural to look for a connection between GPs and low-rank matrix factorisation, which is the motivation and contribution of our work.

To develop a framework for this connection, we first propose a novel approximation scheme for GPs. Our starting point is the Kronecker structure that arises naturally when working with kernels that are products of simpler constituent kernels (say each dependent on one covariate dimension). Coupled with the weight space view of GPs, we can represent a draw from the GP as a product between a random weight tensor and a collection of feature vectors (one for each constituent kernel). The weight tensor can be very large for high dimensional problems, and our proposal is to approximate it using a low-rank Tucker decomposition [Tucker, 1966] instead. This reduces the effective number of parameters that need to be learnt, and forms the link between GPs and matrix factorisation methods in CF. Thus we arrive at our model, the Tucker Gaussian Process (TGP).

We make the following contributions:

- TGP is an elegant and effective method for modelling user/item similarities via kernels to exploit side-information. As far as we know, ours is the first work to use GPs for modelling similarities via side information, with explicit correspondence between similarities and the kernel.

- TGP generalises classical Bayesian Matrix factorisation models [Salakhutdinov and Mnih, 2008b,a], bridging the gap between matrix factorisation methods and GP methods in CF.

- Sub-linear scaling of TGP, achieved by stochastic gradient descent, makes it suitable for CF problems that typically have large data sets infeasible for GPs.

- TGP is applicable to certain regression problems where the regression function is separable in the covariates (i.e. can be written as a product of each covariate). We reason that the Tucker decomposition acts as a regulariser for the GP that

helps control overfitting, and verify that TGP outperforms GPs in generalisation performance for these problems.

Outline of paper We formulate TGP in the general regression setting in Section 3.2, and describe its central application to CF with side information in Section 3.3, followed by related work and discussion in Section 3.4. We present experimental results in Section 3.5 and conclude in Section 3.6.

3.2 Tucker Gaussian Process Regression

3.2.1 Tucker GP Regression

Consider a regression problem with inputs $x_1, \ldots, x_N \in \mathcal{X}$ and corresponding observations $y_1, \ldots y_N \in \mathbb{R}$. We assume $y_i|x_i \sim \mathcal{N}(f(x_i), \sigma^2)$ for some $f : \mathcal{X} \to \mathbb{R}$ and that the observations are independent. The aim is to learn f. One approach is to put a Gaussian Process (GP) prior on f, with zero mean and covariance k. The training then consists of computing the posterior GP. The problem of using this in a CF setting is that training costs $O(N^3)$ operations. N, the number of ratings, usually ranges from a hundred thousand to hundreds of millions in CF, making inference infeasible.

The weight space view of GPs offers a natural way of dealing with the problem: suppose there exists a feature map $\phi : \mathcal{X} \to \mathbb{R}^n$ (where n is the number of features) such that $k(x, x') = \phi(x)^\top \phi(x') \; \forall x, x' \in \mathcal{X}$. Then the GP is equivalent to Bayesian Linear Regression with feature vectors used for each row of the design matrix [Rasmussen and Williams, 2005]:

$$\begin{aligned} y|x, \theta &\overset{ind}{\sim} \mathcal{N}(f(x), \sigma^2), \quad f(x) = \theta^\top \phi(x) \\ \theta &\sim \mathcal{N}(0, I), \quad \theta \in \mathbb{R}^n \end{aligned} \tag{3.1}$$

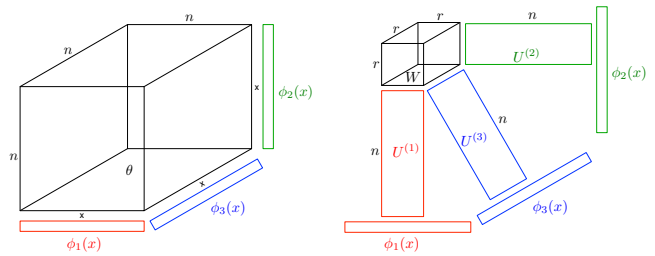
Now training takes $O(Nn^2)$ time, and is scalable for $n \ll N$.

Consider the case of product kernels, where the kernel can be written as follows:

$$k(x_i, x_j) = \prod_{d=1}^{D} k_d(x_i, x_j) \tag{3.2}$$

and suppose there are feature maps $\phi_d : \mathcal{X} \to \mathbb{R}^n$ such that $k_d(x_i, x_j) = \phi_d(x_i)^\top \phi_d(x_j)$. Then we can write $k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$ where $\phi(x) = \otimes_{d=1}^{D} \phi_d(x)$ is the Kronecker product of the ϕ_d. Returning to (3.1):

$$f(x) = \theta^\top \phi(x) = \theta^\top \big(\otimes_{d=1}^{D} \phi_d(x) \big) = \theta \times_{d=1}^{D} \phi_d(x) \tag{3.3}$$



(a) Tensor (b) Tucker

Figure 3.1: Tensor & Tucker Decomposition representation of regression function for $D = 3$.

where θ has been reshaped as a D-dimensional tensor in $\mathbb{R}^{n\times\ldots\times n}$ in the rightmost expression, as in Figure 3.1a. We define the tensor product notation as follows:

$$\theta \times_{d=1}^{D} \phi_d := vec(\theta)^\top \otimes_{d=1}^{D} \phi_d = \sum_{i_1,\ldots,i_D=1}^{n} \theta_{i_1,\ldots,i_D} \prod_{d=1}^{D} (\phi_d)_{i_d}$$

We refer to (3.3) as the *full-rank* model, and use θ as a tensor for the rest of the paper.

This *full-rank* model is problematic in high dimensions: the size of θ grows as n^D, so the function computation become infeasible. Thus we introduce the novel *Tucker Gaussian Process* (TGP) model, where we circumvent this problem by approximating θ using a low-rank Tucker decomposition [Tucker, 1966]. This is defined as a tensor-matrix product between a low rank core tensor $W \in \mathbb{R}^{r\times\ldots\times r}$ of dimension D and matrices $U^{(1)}, \ldots U^{(D)} \in \mathbb{R}^{n\times r}$, as in Figure 3.1b. We denote $\theta \approx W \times_{d=1}^{D} U^{(d)\top}$ where

$$[W \times_{d=1}^{D} U^{(d)\top}]_{i_1,\ldots,i_D} := W \times_{d=1}^{D} U^{(d)}_{i_d} = \sum_{j_1,\ldots,j_D=1}^{r} W_{j_1,\ldots,j_D} \prod_{d=1}^{D} U^{(d)}_{i_d,j_d}.$$

n is the number of features in each dimension and r is the rank. Note that we are free to use a different n and r for each dimension, but assume these are the same as a simplifying assumption. Tucker decomposition is one of the most commonly used tensor decomposition along with PARAFAC decomposition [Bro, 1997], a special case of Tucker decomposition where the core tensor W is diagonal.

We must also place suitable priors on W and $U^{(d)}$ to match the iid $\mathcal{N}(0,1)$ prior on each entry of θ. We place iid priors $\mathcal{N}(0,1)$ on each entry of W, and $\mathcal{N}(0,\sigma_u^2)$ on each entry of $U^{(d)}$. Note that the resulting model is not a GP since the product of Gaussians is not Gaussian, however setting $\sigma_u^2 = \frac{1}{r}$, we match the first two moments of $W \times_{d=1}^{D} U^{(d)\top}$ and θ. We then prove in Appendix 3.7.2 that each entry of $W \times_{d=1}^{D} U^{(d)\top}$ converges in distribution to $\mathcal{N}(0,1)$ as $r \to \infty$.

In summary our TGP regression model approximating data from a GP with product kernel (3.2) and homoscedastic noise is:

$$y|x \overset{ind}{\sim} \mathcal{N}(f(x), \sigma^2),\ f(x) = W \times_{d=1}^{D} \left(U^{(d)\top} \phi_d(x)\right) \tag{3.4}$$

where $\phi_d : \mathcal{X} \to \mathbb{R}^n$ are feature maps such that $k_d(x_i, x_j) = \phi_d(x_i)^\top \phi_d(x_j)$, and we have iid $\mathcal{N}(0,1)$ and $\mathcal{N}(0, \frac{1}{r})$ priors on the entries of W and $U^{(d)}$ respectively.

3.2.2 Choice of Feature Map

So far, we have assumed that the kernels k_d can be written as the inner product of feature vectors: $k_d(x_i, x_j) = \phi_d(x_i)^\top \phi_d(x_j)$. We investigate the situations where this assumption holds. When this doesn't hold, we explore other choices of ϕ that approximate k_d.

Identity features One case where we can write kernels as inner products of features is with identity kernels $k_d(x_i, x_j) = \delta_{ij}$. The features are unit vectors: $\phi_d(x_i) = e_i := (0, \cdots, 0, 1, 0, \cdots)^\top$ with the non-zero at the i^{th} entry, hence $U^{(d)\top} \phi_d(x_i) = U_i^{(d)}$. However this implies $U^{(d)} \in \mathbb{R}^{N \times r}$ (or $\mathbb{R}^{n_d \times r}$ for inputs on a grid of size $n_1 \times \ldots n_D$), so for N (or n_d) too big, computations can become too costly both in time and memory. A workaround is to use feature hashing [Weinberger et al., 2009] to obtain shorter features whose inner products are unbiased estimates of inner products of the original features. This technique can be applied to arbitrary features where the number of features is too large. See Appendix 3.7.3 for details.

We can also deal with cases where the data lies on a grid using Cholesky features, or in the most general case where the data doesn't lie on a grid and k_d cannot be expressed as the inner product of finite feature vectors using Random Fourier features. See Appendix 3.7.5 for details.

3.2.3 Learning

In TGP we would like to learn the posterior distribution of U and W. The simplest and fastest method of learning is Maximum a Posteriori (MAP), whereby we approximate the posterior with point estimates $\hat{U}, \hat{W} = \arg\max_{U,W} p(U, W|y)$. For the optimisation we may use stochastic gradient descent (SGD) to approximate the full gradient, for which we get a time complexity of $O(m(nrD + r^D D))$ operations for computing the stochastic gradient on a mini-batch of size m, which is sublinear in N. See Appendix 3.7.1 for a details.

The problem with a MAP estimate for U, W is that only the posterior mode is used, and the uncertainty encoded in the shape of the posterior distribution is ignored. In a Bayesian setting, we wish to use samples from the posterior and average predictions over samples. For data where we can afford an $O(N)$ runtime, we may use sampling algorithms such as Hamiltonian Monte Carlo (HMC) [Duane et al., 1987, Neal, 2011]. The runtime for each HMC leapfrog step is $O(N(nrD + r^D D))$, the same time complexity as a step of full-batch gradient descent.

3.3 TGP for Collaborative Filtering with Side Information

In order to apply TGP to CF, let us first formulate the problem using GPs. It is natural to model this as a supervised regression problem with $R_{ij} \sim \mathcal{N}(f(u_i, v_j), \sigma^2)$ and prior $f \sim \mathcal{GP}(0, k)$ [Yu et al., 2006]. Note that this is particularly suitable with side information, since kernels can be interpreted as measures of similarity; we can design k to encode similarities between users/movies given by the side information. Hence we may further exploit the use of GPs for addressing this problem. In particular we use a product kernel $k((u_i, v_j), (u_{i'}, v_{j'})) = k_1(u_i, u_{i'})k_2(v_j, v_{j'})$ since we expect similar ratings for two user/movie pairs if the users are similar *and* the movies are similar. When there is no side information, it is sensible to use identity kernels $k_1(u_i, u_{i'}) = \delta_{u_i u_{i'}}$, $k_2(v_j, v_{j'}) = \delta_{v_j v_{j'}}$. i.e. that distinct users and movies are not similar a priori. With side information, we may add on further kernels κ_1, κ_2 modelling similarity between users/movies: $k_1(u_i, u_{i'}) = a_1^2\delta_{u_i u_{i'}} + b_1^2\kappa_1(u_i, u_i')$, $k_2(v_j, v_{j'}) = a_2^2\delta_{v_j v_{j'}} + b_2^2\kappa_2(v_j, v_j')$, where a and b are parameters controlling the extent to which similarity in side information leads to similarity in preference.

However, it is not immediately clear how this single GP framework relates to the matrix factorisation approach. We show that our proposed TGP forms a natural connection between these two approaches, and that we recover classic matrix factorisation models as a special case. To apply TGP, first note that we have $D = 2$, and the Tucker Decomposition is simply a low-rank matrix factorisation. Using the notation U, V instead of $U^{(1)}, U^{(2)}$, we have that $\theta \approx UWV^\top$, hence $f(u_i, v_j) = \phi_1(u_i)^\top UW(\phi_2(v_j)^\top V)^\top$. With the identity kernel, we have unit vector features $\phi_1(u_i) = e_i \in \mathbb{R}^{n_1}$ and $\phi_2(v_j) = e_j \in \mathbb{R}^{n_2}$. TGP therefore simplifies to:

$$R_{ij} \overset{ind}{\sim} \mathcal{N}(f(u_i, v_j), \sigma^2), \quad f(u_i, v_j) = U_i^\top W V_j \tag{3.5}$$

with iid $\mathcal{N}(0, \sigma_u^2)$ priors on each entry of U, V where U_i, V_j are column vectors representing the i^{th} and j^{th} row of U and V respectively. Note that with $W = I$ fixed, we recover Probabilistic Matrix Factorization (PMF) [Salakhutdinov and Mnih, 2008a], a particularly effective Bayesian model in the matrix factorization framework.

An extension is Bayesian PMF (BPMF) [Salakhutdinov and Mnih, 2008b] where priors are placed on the prior mean and covariance of U_i, V_j. Should we decide to learn W in TGP, interesting parallels arise between our model and BPMF. Observe from the following that learning W can be a proxy for learning the prior mean and covariance of U and V, as is done in the BPMF model:

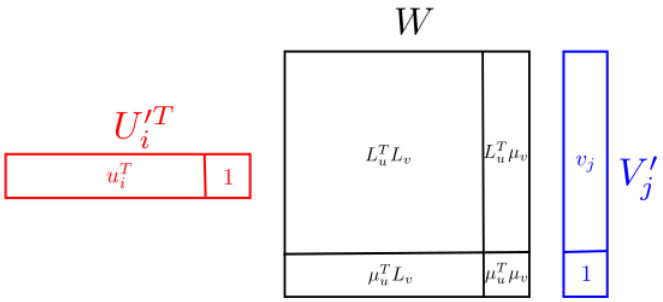


Figure 3.2: Bayesian PMF reparametrised.

$$U_i \sim \mathcal{N}(\mu_u, \Lambda_u), V_j \sim \mathcal{N}(\mu_v, \Lambda_v) \Rightarrow U_i = \mu_u + L_u u_i, V_j = \mu_v + L_v v_j$$
$$\text{where } u_i, v_j \sim \mathcal{N}(0, I), \Lambda_u = L_u L_u^\top, \Lambda_v = L_v L_v^\top$$
$$\Rightarrow U_i^\top V_j = \mu_u^\top \mu_v + \mu_u^\top L_v v_j + u_i^\top L_u^\top \mu_v + u_i^\top L_u^\top L_v v_j = U_i'^\top W V_j'$$

where $U_i'^\top = [u_i^\top, 1]$, $W = [L_u^\top L_v, L_u^\top \mu_v; \mu_u^\top L_v, \mu_u^\top \mu_v]$, $V_j' = [v_j; 1]$, as displayed in Figure 3.2. So a full W with standard iid Gaussian priors on U, V can capture the effects of modelling U, V with non-zero means and full covariances for each row of U, V, as in BPMF.

Returning to the case with side information, suppose it is given in the form of vectors $\omega_1(u_i), \omega_2(v_j)$, and that we expect users/movies with similar ω to show similar preferences/be preferred by similar users. For example we can encode the user age into ω_1 and the movie genre into ω_2 and define $\kappa_1(u_i, u_i') = \omega_1(u_i)^\top \omega_1(u_{i'}), \kappa_2(v_j, v_j') = \omega_2(v_j)^\top \omega_2(v_{j'})$. The feature vector is now $\phi_d(u_i) = [a_d e_i^\top, b_d \omega_d(u_i)^\top]^\top$ for $d = 1, 2$, and we have $f(u_i, v_j) = \phi_1(u_i)^\top U W V^\top \phi_2(v_j)$.

3.4 Related Work and Discussion

Modelling data in the form of matrices and tensors has been studied in the field of multi-way data analysis and relational learning. The key idea here is to factorise the data tensor, with two notable forms of factorisation: PARAFAC [Bro, 1997] and Tucker [Tucker, 1966]. There are a few works in these domains that relate to GPs. InfTucker [Xu et al., 2012] uses the Tucker decomposition directly on the data tensor, and use a non-linear transformation of the parameters $U^{(d)}$ for the regression function, contrary to TGP which is linear in the parameters. DinTucker [Zhe et al., 2016a] tries to scale up InfTucker by splitting up the observed tensor into subarrays. [Imaizumi and Hayashi, 2016] motivate their model using the Parafac decomposition instead of Tucker, expressing the regression function as a sum of products of local functions. These local functions are each modelled by GPs. However the TGP is motivated from a single GP on the input space. [Suzuki et al., 2016] again model the regression function as a sum of products of local functions, which live in the RKHS of some kernel, analogous to the feature maps in TGP. However there is no mention of GPs or how their model relates to low-rank tensor decomposition. [Zhao et al., 2013] deals with the classification problem where each input is a tensor, so there is one label per tensor. They define a GP over the space of tensors. It is unclear whether they actually use low rank tensor decomposition. For TGP we deal with regression, work in the setting where each element of the data tensor corresponds to a response, and apply Tucker decomposition to the parameters. [Zhe et al., 2016b] define a GP over the parameter space whereas TGP is a multilinear expression in the parameters and feature maps, approximating a GP over the input space. In short, these models make different assumptions to TGP, and thus are useful for different CF applications – none use side information (it is unclear how this would be possible given their model assumptions) and do not relate to the Bayesian matrix factorisation literature.

There are closer connections between our model and the Stochastic Relational Model [Yu et al., 2006] in relational learning. It is a special case of our model with $W = I$ and $D = 2$. The key differences lie in the inference: we use features to build on the weight-space view of GPs, whereas Yu et al. [2006] work with GPs in the function-space view. This complicates learning for kernels which cannot be expressed as an inner product of features; the authors resort to the Laplace approximation for finding maximum likelihood estimates of parameters. For such kernels we use random feature maps (see Appendix 3.7.5), making learning simple and more computationally efficient.

In the domain of matrix factorisation, Lawrence and Urtasun [2009] use a GP Latent Variable Model (GP-LVM) [Lawrence, 2004]. They learn a latent vector for each movie, and pass it through a zero-mean GP with a squared exponential (SE) kernel, with one GP per user. In TGP we use one GP for all users and items. For a CF application, they incorporate side information about movies by taking the product of these kernels with a SE kernel in the movie features. Our model is more flexible in that we can take into account both user and item similarities simultaneously.

From the perspective of GP regression, we analyse the regression function of TGP to understand the regression problems for which it will be effective. Recall that the regression function $f(x)$ in (3.4) can be seen as $W \times_{d=1}^{D} \psi_d(x)$, where $\psi_d(x) = U^{(d)\top}\phi_d(x)$ are lower-dimensional features in $\mathbb{R}^r$ (i.e. the $U^{(d)}$ multiplied by $\phi_d(x)$ in Figure 3.1b). With this new formulation, we have:

$$f(x) = W \times_{d=1}^{D} \phi_d(x) = \sum_{i_1,\ldots,i_D=1}^{r} W_{i_1\ldots i_D} \prod_{d=1}^{D} (\psi_d(x))_{i_d} \tag{3.6}$$

Hence learning W and $\left(U^{(d)}\right)_{d=1}^{D}$ can be interpreted as learning features ψ_d as well as their weights for the regression function, i.e. learning a linear combination of products of these features. In the case where $\psi_d(x)$ is only a function of the d^{th} dimension of x, each $\prod_{d=1}^{D}(\psi_d(x))_{i_d}$ is separable in the dimensions. Modelling data with sums of separable functions has been studied in Beylkin et al. [2009], and its effectiveness for regression is shown by promising results on various synthetic and real data. Such additive models arise frequently in the context of ensemble learning, such as *boosting* and BART [Chipman et al., 2010], where a linear combination of many weak learners is used to build a single strong learner. We may interpret our model in this framework where $\prod_{d=1}^{D}(\psi_d(x))_{i_d}$ are the weak learners that share parameters, and $W_{i_1\ldots i_D}$ are the corresponding weights.

With this alternative interpretation in mind, we may expect TGP to perform well in cases where the data displays an additive structure, with the additive components arising from a product of features on each dimension. Hence we interpret TGP as a modified GP where the approximation acts as a regulariser towards such simpler functions, which can actually lead to enhanced generalisation performance by controlling overfitting. We thus compare its performance to GPs on spatio-temporal data sets where it is reasonable to expect separability in longitude and latitude, or in time and space.

Based on Section 3.2.2, we also see that our model is particularly well-suited to modelling grid-structured data. The difference between our model and that in Saatçi [2011] is that we have Kronecker structure in the features ϕ, whereas they exploit Kronecker structure on the data. Moreover, our model can deal with data not on a grid, as well as data on a grid with many missing observations, since observations are not needed for constructing the features.

Going back to CF, recall from Section 3.3 that the low-rank matrix factorisation model has $R_{ij} \approx \sum_k U_{ik} V_{jk}$, a sum of a product of parameters (features) in each dimension. TGP generalises this to modelling a linear combination of products of features, hence we may expect it to perform well for this task. Also note the grid structure, since users and items are categorical variables.

3.5 Experimental Results

Collaborative Filtering We use the MovieLens 100K data[1], which consists of 100,000 ratings in $\{1, \ldots, 5\}$ from 943 users on 1682 movies. User age, gender and occupation are given, as well as the genre of the movies. We represent this side information with binary vectors for $\omega_1(u_i), \omega_2(v_j)$ and use the formulation in (3.7) in Appendix 3.7.6. We bin the age into five categories, and there are 20 occupations and 18 genres. Thus $\omega_1(u_i) \in \mathbb{R}^{5+2+20}$ has 3 non-zero entries, one for each feature, and $\omega_2(v_j) \in \mathbb{R}^{18}$ can have multiple non-zero entries since each movie can belong to many genres. We report the mean and standard deviation of the test RMSE on the five 80:20 train test splits that come with the data, as it will offer a sensible means of comparison with other algorithms. N is too large for HMC, hence we use SGD to obtain MAP estimates for the parameters, and compare different configurations: learning W/fixing it to be the identity and using/not using side information, along with BPMF initialised by PMF[2]. We use mini-batches of size 100, and set $r = 15$ for all models as it gives best results for PMF and BPMF. We used a grid search and cross-validation for tuning hyperparameters, the recommended method in big N settings where the number of hyperparameters is not too large. SGD was not so sensitive to mini-batch size, and finding the range of hyperparameters was straightforward. See Appendix 3.7.6 for details.

[1]Obtained from `http://grouplens.org/datasets/movielens/100k/`

[2]Code obtained from `http://www.cs.toronto.edu/~rsalakhu/BPMF.html`

Table 3.1: Test RMSE on MovieLens100K.

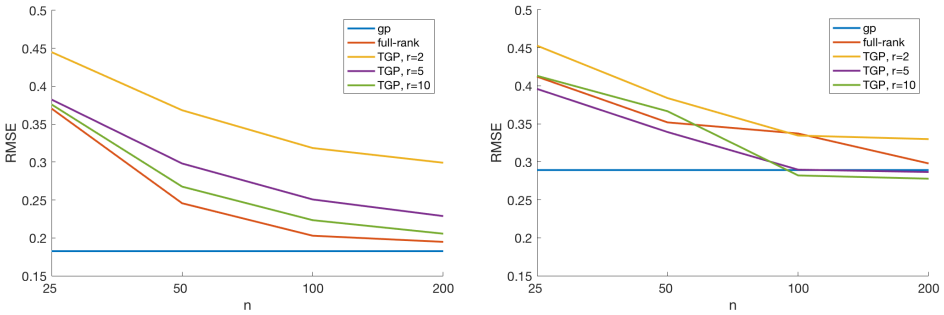
Model	**Test RMSE**
BPMF	0.9024 ± 0.0050
TGP, $W = I$ (PMF)	0.9395 ± 0.0115
TGP, learn W	0.9270 ± 0.0097
TGP, $W = I$, side-info	0.9014 ± 0.0061
TGP, learn W, side-info	$\mathbf{0.8995 \pm 0.0062}$

From Table 3.1 it is evident that TGP makes good use of side information, since the RMSE decreases with side information. Learning W instead of fixing it helps predictive performance, but does not perform as well as BPMF. One reason is that our Gaussian prior on W is not equivalent to the Gaussian-Wishart priors on the mean and variance of U_i, V_j in BPMF. Another reason is that we are resorting to a MAP estimate. If we can instead sample from the posterior and average predictions over these samples, we expect enhanced predictions. However, note that using TGP with side information and learning W, we are able to get comparable/superior results to BPMF, even with a MAP estimate. We expect further improvements not only with sampling but also by using more sophisticated kernels that make better use of the side information; for example, using different hyperparameter coefficients for the different types of features. In so far as comparison was possible, these numbers are comparable to state-of-the-art algorithms in Section 3.4. A direct comparison was not possible as each use different methods for evaluation.

Regression on spatial data We use the California house prices data from the 1990 census[3], which consists of average house prices for 20,640 different locations in California. We only use the covariates longitude and latitude, and whiten them along with log-transformed house prices to each have zero mean and unit variance. We chose this data set as spatial data sometimes exhibit separability in the different dimensions. Moreover the data is clustered in urban areas, hence an additive model with each component describing different sections of California may be desirable. Using a random 50:50 train test split, we report the RMSE of the model on the training set and test set after training. We first fit a GP to the data with a squared exponential (SE) kernel on each dimension using the GPML toolbox [Rasmussen and Nickisch, 2010], optimising the hyperparameters by type-II maximum likelihood. Then using these hyperparameters we generate RFF for ϕ. See Appendix 3.7.4 and 3.7.5 for details. We implemented both the *full-rank* model and TGP with $n = 25, 50, 100, 200$ on

[3]Obtained from `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/cadata`

Stan [Stan Development Team, 2012], which uses HMC with the No-U-Turn Sampler (NUTS) [Hoffman and Gelman, 2014] for inference. Note that for both models n refers to the length of features $\phi_d(x)$.

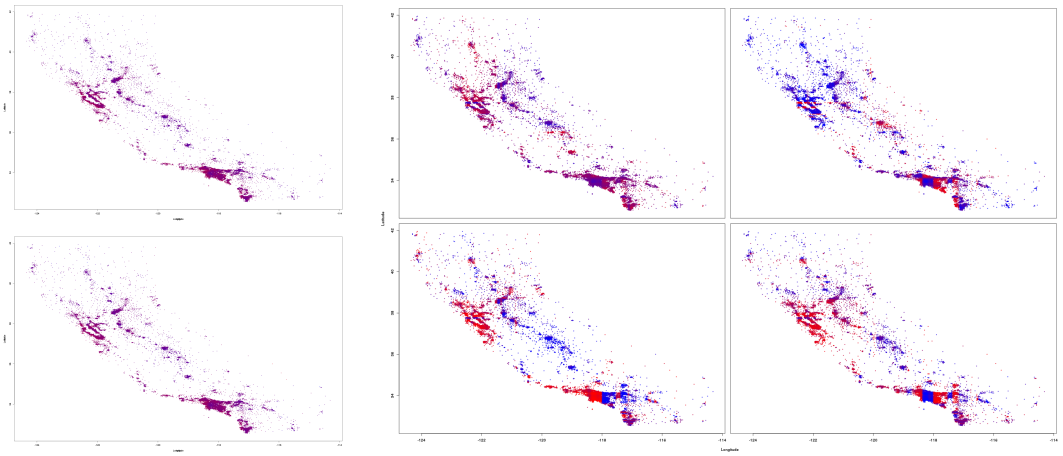


(a) Train RMSE (b) Test RMSE

Figure 3.3: RMSE for GP, *full-rank*, and TGP for $r = 2, 5, 10$ for $n = 25, 50, 100, 200$ on the California House Price data.

For TGP, we use 300 warmup iterations and a further 300 samples on 4 different chains, and use the mean prediction across the samples. For *full-rank*, we take the same number of samples and chains, but only use 50 warmup draws as we diagnosed that convergence was reached by this point (looking at the Gelman-Rubin statistic [Gelman and Rubin, 1992] and effective sample size). The convergence statistics for TGP are in Appendix 3.7.7. We can see from Figure 3.3 that some TGP models give lower test RMSE and higher train RMSE than the GP and the *full-rank* model. In fact TGP with $r = 5$ consistently shows higher predictive performance than *full-rank* for all values of n, and for $n \geq 100$ TGP with $r = 10$ outperforms GP. This indicates that TGP is an effective regulariser towards simpler regression functions, namely a linear combination of separable functions. We expect bigger gains for TGP with more warmup iterations, since the convergence diagnostics suggest that TGP hasn't quite fully mixed by 300 iterations.

We further investigate the predictions of TGP by analysing the additive components in the prediction for $r = 2$. We see in Figure 3.4 that the components are quite different. The upper two components show complementary predictions in the Bay area (North-West) and the central area, whereas the bottom two show complementary predictions in the Los Angeles area (South-East). This confirms the hypothesis that the different additive components will learn different sections of the data. See Appendix 3.7.7 for zoomed in plots, and Appendix 3.7.8 for experimental results on spatio-temporal data with grid structure.



(a) Top: Heatmap of true log house price values. Bottom: TGP predictions for $r = 2, n = 200$.

(b) Heatmap showing the four additive components (summands in (3.6)) of predictions for TGP with $r = 2, n = 200$.

Figure 3.4: We only use the last sample in the Markov Chain to get a better indication of the structure. Red indicates high log price and blue indicates low, and the same colour scheme is applied to all four subplots. To accentuate the differences in the predictive values, we colour values by the percentile they belong to instead of a uniform colouring. See Appendix 3.7.7 for the uniform colouring.

3.6 Conclusion

We have introduced the Tucker Gaussian Process (TGP), a regression model that regularises a GP towards simpler regression functions, in particular a linear combination of separable functions. We motivate it as a solution to Collaborative Filtering (CF), by using feature maps and a low-rank Tucker decomposition on the parameters in the weight-space view of GPs. In particular, we have highlighted the effectiveness of TGP in CF with side-information, a domain where outputs can be effectively modelled as a linear combination of functions separable in the covariates. We believe that this is the largest contribution of our paper: after showing that PMF and BPMF are special cases of the TGP, we extended it to exploit the user/item side information for better predictions; the kernel of the GP we approximate can be designed to encode similarities between different users and items, a particularly neat and natural method for modelling similarity. In doing so, we bring together matrix factorisation methods and GP methods in CF, as well as scaling up GP methods in CF. We confirm experimentally that side information enhances the predictive performance of TGP in collaborative filtering.

We have also shown that for problems where one might expect separability in the covariates such as prediction for spatio-temporal data sets, the TGP effectively controls overfitting and outperforms GPs in prediction. We also point out that exact Cholesky features can be used with TGP in the case of grid-structured data, and random feature maps can be used for arbitrary kernels.

Acknowledgements

HK, XL, SF and YWT's research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 617071.

3.7 Supplementary Material

3.7.1 Learning Algorithms for TGP

We give detailed derivations of various inference algorithms for the TGP. We have a set of N observations $y_i \in \mathbb{R}$ corresponding to a set of inputs $x_i \in \mathcal{X}$, and we wish to regress $y = (y_i)_{i=1}^N$ on $X = (x_i)_{i=1}^N$. We assume that the data generating mechanism takes the form

$$y = f(X) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma^2 I_N)$$

where $f(X) = (f(x_i))_{i=1}^N \in \mathbb{R}^N$ and also that the regression function takes the following form

$$f(x) = w^\top \otimes_{d=1}^D \left(U^{(d)\top} \phi_d(x) \right)$$

where

- $W \in \mathbb{R}^{r \times \ldots \times r}$ is a D-dimensional tensor whose entries are iid $\mathcal{N}(0, \sigma_w^2)$
- $w = vec(W)$ is the vector obtained when flattening tensor W, such that $W \times_{d=1}^D v_d = w^\top \otimes_{d=1}^D v_d \ \forall v_d \in \mathbb{R}^r$
- $(\phi_d(x))_{d=1}^D$ are the features in $\mathbb{R}^n$ extracted from x
- $(U^{(d)})_{d=1}^D$ are a set of real $n \times r$ matrices with $U_{jl}^{(d)} \overset{iid}{\sim} \mathcal{N}(0, \sigma_u^2)$

We assume $n > r$, and wish to learn w and the $U^{(d)}$ from the data.

Note from the second point that $\nabla_w\big(W \times_{d=1}^D v_d\big) = \otimes_{d=1}^D v_d$. For D=2 for example, if $g(U) = s^\top U t$ for some matrix U and vectors s, t, then $\nabla_u g(U) = s \otimes t$ where $u = vec(U)$.

First we give the complexity for calculating $f(x)$. Computing $\psi_d(x_i) = U^{(d)\top}\phi_d(x_i)\ \forall d$ requires $O(nrD)$ time, then $w^\top \otimes_{d=1}^D \psi_d(x_i)$ takes $O(r^D)$ time. So time for a prediction given ϕ, U, w takes $O(nrD + r^D)$.

The quantity of interest for MAP and HMC is the log joint distribution $p(y, U, w) = p(y|U, w)p(U)p(w)$. In full this is:

$$\log p(y|U,w)+\log p(U)+\log p(w) = -\frac{1}{2\sigma^2}\sum_{i=1}^N (y_i - f(x_i))^2 - \frac{1}{2\sigma_u^2}\sum_{k=1}^D tr(U^{(k)T}U^{(k)}) - \frac{1}{2\sigma_w^2} w^\top w$$

This has the following derivatives:

$$\begin{aligned}
\nabla_w \log p(w) &= -w \\
\nabla_{U^{(k)}} \log p(U) &= -rU^{(k)} \\
\nabla_w \log p(y_i|U,w) &= \frac{1}{\sigma^2}(y_i - f(x_i)) \otimes_{d=1}^D \psi_d(x_i) \\
\nabla_{u^{(k)}} \log p(y_i|U,w) &= \frac{1}{\sigma^2}(y_i - f(x_i))\phi_k(x_i) \otimes \big(W \times_{d\neq k} \psi_d(x_i)\big)
\end{aligned}$$

with the following definitions:

- $u^{(k)} = vec(U^{(k)}) \in \mathbb{R}^{nr}$
- $(W \times_{d\neq k} v_d)_l := W \times_{d=1}^D v'_d$ where $v'_d = v_d$ for $d \neq l$ and $v'_l = e_l \in \mathbb{R}^r$, the unit vector with non-zero at the l^{th} entry.

The last derivative holds since $f(x) = \phi_k(x)^\top U^{(k)}(W \times_{d\neq k} \psi_d(x_i))$. Computing $W \times_{d\neq k} \psi_d(x_i)$ takes $O(r^D)$ for each k, hence $O(r^D D)\ \forall k$. So we have that using mini-batches $\{x_{t1}, \ldots, x_{tm}\}$ for SGD, we have the following updates for MAP:

$$w \leftarrow w + \frac{\epsilon^w}{2}\left(\nabla_w \log p(w_t) + \frac{N}{m}\sum_{i=1}^m \nabla_w \log p(y_{ti}|x_{ti}, w, U)\right)$$

$$u^{(k)} \leftarrow u^{(k)} + \frac{\epsilon^k}{2}\left(\nabla_{u^{(k)}} \log p(U) + \frac{N}{m}\sum_{i=1}^m \nabla_{u^{(k)}} \log p(y_{ti}|x_{ti}, w, U)\right)$$

with time complexity $O(m(nrD + r^D D))$.

Gathering the parameters into a vector $\theta = (w, U^{(1)}, \ldots, U^{(k)})$, the HMC algorithm runs as follows:

1. Initialise MC by drawing $\theta_0 = (w_0, U_0^{(1)}, ..., U_0^{(D)})$ from its prior distribution.

2. For $t = 0, ..., T$:

   (a) Initialise $p \sim \mathcal{N}(0, I_Q)$, $V^{(k)} \sim \mathcal{N}(0, I_{n\times r}) \quad \forall k$,
   $H_t \leftarrow -\log p(w_t) - \sum_{i=1}^N \log p(y_i|x_i, \theta_t) + \frac{1}{2}\sum_{k=1}^D tr(V^{(k)T}V^{(k)}) + \frac{1}{2}p^T p$
   $\theta = (w, U^{(1)}, ...U^{(D)}) \leftarrow \theta_t = (w_t, U_t^{(1)}, ...U_t^{(D)})$

   (b) For $l = 1, ..., L$:

      i. $p \leftarrow p + \frac{\epsilon_t^w}{2}\Big(\nabla_w \log p(w_t) + \sum_{i=1}^N \nabla_w \log p(y_i|x_i, \theta_t)\Big)$
      For $k = 1, ..., D$:
      $V^{(k)} \leftarrow V^{(k)} + \frac{\epsilon_t^k}{2}\Big(\sum_{i=1}^N \nabla_{U^{(k)}} \log p(y_i|x_i, \theta)\Big)$

      ii. $w \leftarrow w + \epsilon_t^w p$
      For $k = 1, ..., D$:
      $u^{(k)} \leftarrow u^{(k)} + \epsilon_t^w V^{(k)}$

      iii. same as i.

   (c) $H^* \leftarrow -\log p(w) - \sum_{i=1}^N \log p(y_i|x_i, \theta) + \frac{1}{2}\sum_{k=1}^D tr(V^{(k)T}V^{(k)}) + \frac{1}{2}p^T p$
   $u \sim Unif[0, 1]$
   If $u \leq \exp(H_t - H^*)$
   $\theta_{t+1} = (w_{t+1}, U_{t+1}^{(1)}, ..., U_{t+1}^{(D)}) \leftarrow \theta = (w, U^{(1)}, ..., U^{(D)})$
   else
   $\theta_{t+1} \leftarrow \theta_t$

From previous computations, it is easy to see that each update requires $O(LN(nrD + r^D D))$ operations.

3.7.2 Elementwise convergence of TGP to $\mathcal{N}(0, 1)$

Definition 3.7.1. ***Martingale Difference Sequence*** *A martingale difference sequence with respect to a filtration $(\mathcal{F}_p)_{p\in\{0,1,...,r\}}$ is a real-valued sequence of random variables $X_1, \ldots, X_r$ that satisfies:*

1. X_p *is* $\mathcal{F}_p$ *measurable*
2. $\mathbb{E}(|X_p|) < \infty$
3. $\mathbb{E}(X_p|\mathcal{F}_{p-1}) = 0$ *a.s.*

for all $p \in \{1, \ldots, r\}$.

Theorem 2 (**Martingale Central Limit Theorem** [Hall and Heyde, 2014])**.** *Let* $X = \{X_1, \ldots, X_r\}$ *be a sequence of random variables satisfying the following conditions:*

1. X *is a martingale difference sequence with respect to filtration* $(\mathcal{F}_p)_{p\in\{0,1,\ldots,r\}}$
2. $\sum_{p=1}^{r} \mathbb{E}(X_p^2|\mathcal{F}_{p-1}) \xrightarrow{p} 1$ *as* $r \to \infty$.
3. $\sum_{p=1}^{r} \mathbb{E}(X_p^2 \mathbb{I}(|X_p| > \epsilon)|\mathcal{F}_{p-1}) \xrightarrow{p} 0$ *as* $r \to \infty$ $\forall \epsilon > 0$.

Then the sums $S_r = \sum_{p=1}^{r} X_p \xrightarrow{d} \mathcal{N}(0, 1)$ *as* $r \to \infty$.

Proposition 1. *Let* n *by* r *matrices* $U^{(d)} \overset{iid}{\sim} \mathcal{N}(0, \frac{1}{r}I)$ *for* $d = 1, \ldots, D$, *and let* $W \sim \mathcal{N}(0, I)$ *where* $W \in \mathbb{R}^{r\times\ldots\times r}$ *is a D-dimensional tensor. Then each element of* $W \times_{d=1}^{D} U^{(d)\top}$ *converges in distribution to* $\mathcal{N}(0, 1)$ *as* $r \to \infty$.

Proof. Suppose first that $D = 2$. It suffices to show that

$$u, v \in \mathbb{R}^r, W \in \mathbb{R}^{r\times r}, u, v \overset{iid}{\sim} \mathcal{N}(0, I), W \sim \mathcal{N}(0, I)$$
$$\Rightarrow u^\top W v \xrightarrow{d} \mathcal{N}(0, 1) \text{ as } r \to \infty$$

We define for each $r \in \mathbb{N}$:

$S_0 = 0$

$S_p := \sum_{i,j=1}^{p} u_i W_{ij} v_j$

$X_p := S_p - S_{p-1} = u_p W_{pp} v_p + \sum_{i=1}^{p-1} u_p W_{pi} v_i + u_i W_{ip} v_p$

$\mathcal{F}_0 := \{\emptyset, \Omega\}$ where Ω is the sample space for the RVs u, v, W

$\mathcal{F}_p := \sigma(u_i, v_j, W_{ij})_{i,j=1}^{p}$, the sigma algebra generated by these random variables

for $p \in \{1, \ldots, r\}$

So we have that $S_r = u^\top W v$, hence it suffices to check conditions 1,2,3 in the Martingale CLT.
We first show 1, that X is a martingale difference sequence. It is clear that X_p is $\mathcal{F}_p$ measurable by definition of $\mathcal{F}_p$. To show that X is integrable, we have:

$$\begin{aligned}\mathbb{E}(|X_p|) &\leq \mathbb{E}(|u_p W_{pp} v_p|) + \sum_{i=1}^{p-1} \mathbb{E}(|u_p W_{pi} v_i|) + \mathbb{E}(|u_i W_{ip} v_p|) \\ &\leq \sqrt{\mathbb{E}(u_p^2 W_{pp}^2 v_p^2)} + \sum_{i=1}^{p-1} \sqrt{\mathbb{E}(u_p^2 W_{pi}^2 v_i^2)} + \sqrt{\mathbb{E}(u_i^2 W_{ip}^2 v_p^2)} \\ &= \frac{1}{r} + (p-1)\left(\frac{1}{r} + \frac{1}{r}\right) < \infty\end{aligned}$$

by the inequality $\mathbb{E}(|X|)^2 \leq \mathbb{E}(X^2)$ (shown using convexity of $g : x \to x^2$ and Jensen's inequality) and independence of u, v, W. Also we have:

$$\begin{aligned}\mathbb{E}(X_p|\mathcal{F}_{p-1}) &= \mathbb{E}(u_p W_{pp} v_p) + \sum_{i=1}^{p-1} \mathbb{E}(u_p W_{pi}) v_i + u_i \mathbb{E}(W_{ip} v_p) \\ &= 0\end{aligned}$$

since u_p, v_p, W_{pi}, W_{ip} are independent of $\mathcal{F}_{p-1}$ and have zero mean. Hence X forms a martingale difference sequence.

To verify the next two conditions, we first prove a lemma that will help us do so. This is the generalisation of Chebyshev's inequality to higher moments:

Lemma 3. *Suppose X is a random variable with bounded n^{th} moment for some $n \in \mathbb{N}$. Then*
$\mathbb{P}(|X - \mathbb{E}(X)| > \epsilon) \leq \frac{\mathbb{E}(|X - \mathbb{E}(X)|^n)}{\epsilon^n} \ \forall \epsilon > 0.$

Proof. Without loss of generality, assume $\mathbb{E}(X) = 0$. Then

$$\mathbb{P}(|X| > \epsilon) = \mathbb{E}[\mathbb{I}(|X| > \epsilon)] = \frac{1}{\epsilon^n}\mathbb{E}[\epsilon^n \mathbb{I}(|X| > \epsilon)] \leq \frac{1}{\epsilon^n}\mathbb{E}[|X|^n \mathbb{I}(|X| > \epsilon)] \leq \frac{\mathbb{E}[|X|^n]}{\epsilon^n}$$

□

Note Lemma 3 shows that convergence in L^n implies convergence in probability. So to show conditions 2 and 3 of the martingale CLT, it suffices to show that the expectations of the quantities on the left hand sides converge to the right hand side as scalars:

2'. $\sum_{p=1}^{r} \mathbb{E}(X_p^2) \to 1$ as $r \to \infty$.

3'. $\sum_{p=1}^{r} \mathbb{E}(X_p^2 \mathbb{I}(|X_p| > \epsilon)) \to 0$ as $r \to \infty$ $\forall \epsilon > 0$.

Let us show 2'. In $\mathbb{E}(X_p^2)$, note that all cross terms in $\mathbb{E}(X_p^2)$ cancel since all terms have mean 0. So we have:

$$\begin{aligned}
\mathbb{E}(X_p^2) &= \mathbb{E}(u_p^2 W_{pp}^2 v_p^2) + \sum_{i=1}^{p-1} \mathbb{E}(u_p^2 W_{pi}^2 v_i^2) + \mathbb{E}(u_i^2 W_{ip}^2 v_p^2) \\
&= \frac{1}{r^2} + (p-1)\left(\frac{1}{r^2} + \frac{1}{r^2}\right) = \frac{2p-1}{r^2} \\
\Rightarrow \sum_{p=1}^{r} \mathbb{E}(X_p^2) &= \frac{2}{r^2} \sum_{p=1}^{r} p - r \cdot \frac{1}{r^2} = \frac{2}{r^2} \frac{(r+1)r}{2} - \frac{1}{r} = 1
\end{aligned}$$

To show 3', we first note that for a random variable X,

$$\int_{\delta}^{\infty} \mathbb{I}(X > t) dt = (X - \delta)\mathbb{I}(X > \delta) \text{ for } \delta \in \mathbb{R}$$

Setting $X = X_p^2, \delta = \epsilon^2$ and rearranging we have:

$$\begin{aligned}
X_p^2 \mathbb{I}(|X_p| > \epsilon) = X_p^2 \mathbb{I}(X_p^2 > \epsilon^2) &= \epsilon^2 \mathbb{I}(X_p^2 > \epsilon^2) + \int_{\epsilon^2}^{\infty} \mathbb{I}(X_p^2 > t) dt \\
&= \epsilon^2 \mathbb{I}(|X_p| > \epsilon) + \int_{\epsilon}^{\infty} 2s\mathbb{I}(|X_p| > s) ds \text{ by change of variables } t = s^2 \\
\Rightarrow \mathbb{E}[X_p^2 \mathbb{I}(|X_p| > \epsilon)] &= \epsilon^2 \mathbb{P}(|X_p| > \epsilon) + \int_{\epsilon}^{\infty} 2s\mathbb{P}(|X_p| > s) ds
\end{aligned}$$

Now we would like to use Lemma 3 to upper bound the right hand side. Note we want to use even n such that $\mathbb{E}[|X|^n] = \mathbb{E}(X^n)$, since we know how to compute $\mathbb{E}(X_p^n)$ but not $\mathbb{E}[|X_p|^n]$. Also note that $\mathbb{P}(|X_p| > s)$ can be bounded by $\frac{\mathbb{E}(X_p^n)}{s^n}$. So we want $n > 2$ for the bound on the integral to become finite. Hence we use $n = 4$, and show that $\mathbb{E}(X_p^4)$ is sufficiently small so that even when we sum over $p = 1, \ldots, r$, we have that the upper bound tends to 0 as $r \to \infty$. First we compute $\mathbb{E}(X_p^4)$. Note from the multinomial theorem:

$$(x_1 + x_2 + \cdots + x_m)^n = \sum_{k_1 + k_2 + \cdots + k_m = n} \binom{n}{k_1, k_2, \ldots, k_m} \prod_{1 \le t \le m} x_t^{k_t}$$

where

$$\binom{n}{k_1, k_2, \ldots, k_m} = \frac{n!}{k_1!\, k_2! \cdots k_m!}$$

Applying this to X_p^4 and taking the expectation, we see that the only cross terms that survive are products of even powers of the terms, namely where two of the k_i are 2

and the rest are 0.
Noting $\binom{n}{2,2} = 6$, and that $\mathbb{E}(X^4) = 3\sigma^4$ for $X \sim N(0, \sigma^2)$ we have:

$$\begin{aligned}
\mathbb{E}[X_p^4] &= \mathbb{E}[u_p^4 W_{pp}^4 v_p^4 + \sum_{i=1}^{p-1} u_p^4 W_{pi}^4 v_i^4 + u_i^4 W_{ip}^4 v_p^4] \\
&+ 6\mathbb{E}\bigg[(u_p^2 W_{pp}^2 v_p^2)\bigg(\sum_{i=1}^{p-1} u_p^2 W_{pi}^2 v_i^2 + u_i^2 W_{ip}^2 v_p^2\bigg)\bigg] \\
&+ 6\mathbb{E}\bigg[\sum_{i\neq j}^{p-1} u_p^2 W_{pi}^2 v_i^2 u_p^2 W_{pj}^2 v_j^2 + u_i^2 W_{ip}^2 v_p^2 u_j^2 W_{jp}^2 v_p^2\bigg] \\
&+ 6\mathbb{E}\bigg[\sum_{i,j=1}^{p-1} u_p^2 W_{pi}^2 v_i^2 u_j^2 W_{jp}^2 v_p^2\bigg] \\
&= (2p-1)\cdot\frac{3}{r^2}\cdot 3\cdot\frac{3}{r^2} + 6\cdot 2(p-1)\cdot\frac{3}{r^2}\cdot\frac{1}{r}\cdot\frac{1}{r} \\
&+ 6\cdot 2\binom{p-1}{2}\frac{3}{r^2}\cdot\frac{1}{r^2} + 6(p-1)^2\frac{1}{r^2}\cdot\frac{1}{r^2} \\
&= \frac{3}{r^4}(8p^2+8p-7)
\end{aligned}$$

So $\mathbb{P}(|X_p| > \epsilon) \leq \frac{3}{\epsilon^4 r^4}(8p^2+8p-7)$. Hence

$$\begin{aligned}
\mathbb{E}[X_p^2 \mathbb{I}(|X_p| > \epsilon)] &\leq \frac{3}{\epsilon^2 r^4}(8p^2+8p-7) + \int_\epsilon^\infty 2s\frac{3}{s^4 r^4}(8p^2+8p-7)ds \\
&= \frac{3}{r^4}(8p^2+8p-7)\bigg(\frac{1}{\epsilon^2} + \int_\epsilon^\infty \frac{2}{s^3}ds\bigg) \\
&= \frac{3C}{r^4}(8p^2+8p-7)
\end{aligned}$$

where $\int_\epsilon^\infty \frac{2}{s^3}ds = C$. So

$$\sum_{p=1}^{r} \mathbb{E}[X_p^2 \mathbb{I}(|X_p| > \epsilon)] \leq \frac{C}{r^4}\sum_{p=1}^{r} 8p^2+8p-7 = O\bigg(\frac{1}{r}\bigg) \to 0 \text{ as } r \to \infty$$

since $\sum_{p=1}^{r} 8p^2+8p-7 = O(r^3)$.
So we have shown conditions 1,2',3', hence by martingale CLT we have that

$$S_r = u^\top W v \xrightarrow{d} \mathcal{N}(0,1) \text{ as } r \to \infty$$

We can prove the claim for $D > 2$ in a similar fashion. □

3.7.3 Feature Hashing

Suppose we have features $\phi(x) \in \mathbb{R}^n$. When n is too large, we may use feature hashing [Weinberger et al., 2009] to reduce the dimensionality of ϕ:

Lemma 4. *Let $h : \{1, \ldots, n\} \to \{1, \ldots, m\}$ be a hash function for $m \ll n$. i.e. $\mathbb{P}(h(i) = j) = \frac{1}{m}\ \forall j \in \{1, \ldots, m\}$. Also let $\xi : \{1, \ldots, n\} \to \{\pm 1\}$ be a hash function. Define $\bar{\phi}(x) \in \mathbb{R}^m$ as follows: $\bar{\phi}_j(x) = \sum_{i:h(i)=j} \xi(i)\phi_i(x)$. Then $\mathbb{E}[\bar{\phi}(x)^\top \bar{\phi}(x')] = \phi(x)^\top \phi(x')$, $Var[\bar{\phi}(x)^\top \bar{\phi}(x')] = O(\frac{1}{m})$.*

3.7.4 Random Fourier Features

Theorem 3 (Bochner's Theorem[Rudin, 1964])**.** *A stationary kernel k(d) is positive definite if and only if k(d) is the Fourier transform of a non-negative measure.*

For RFF the kernel can be approximated by the inner product of random features given by samples from its spectral density, in a Monte Carlo approximation, as follows:

$$\begin{aligned} k(x-y) = \int_{\mathbb{R}^D} e^{iv^T(x-y)} d\mathbb{P}(v) \propto \int_{\mathbb{R}^D} p(v) e^{iv^T(x-y)} dv &= \mathbb{E}_{p(v)}[e^{iv^T x}(e^{iv^T y})^*] \\ &= \mathbb{E}_{p(v)}[Re(e^{iv^T x}(e^{iv^T y})^*)] \\ &\approx \frac{1}{n}\sum_{k=1}^{n} Re(e^{iv_k{}^T x}(e^{iv_k{}^T y})^*) \\ &= \mathbb{E}_b[\phi(x)^T \phi(y)] \end{aligned}$$

where $\phi(x) = \sqrt{\frac{2}{n}}(cos(v_1{}^T x + b_1), \ldots, cos(v_m{}^T x + b_n))$ with spectral frequencies v_k iid samples from $p(v)$ and b_k iid samples from $U[0, 2\pi]$.
For a one dimensional squared exponential kernel $k(x, y) = \sigma_f^2 \exp\Big(-\frac{(x-y)^2}{2l^2}\Big)$, the spectral density is $\mathcal{N}(0, l^{-2})$. So we use features $\phi(x) = \sigma_f \sqrt{\frac{2}{n}}(cos(v_1{}^T x + b_1), \ldots, cos(v_m{}^T x + b_n))$ where v_k iid samples from $\mathcal{N}(0, l^{-2})$ and b_k iid samples from $U[0, 2\pi]$.

3.7.5 Choice of Feature Map

Cholesky features Consider data with inputs lying on a D-dimensional grid: $x_i \in \mathcal{X} = \times_{d=1}^{D} X^{(d)}$, $|X^{(d)}| = n_d$ finite, where $k_d(x_i, x_j)$ only depends on the values that x_i, x_j take in $X^{(d)}$. The $X^{(d)}$ can be, for example, a finite set of points in Euclidean space, or the set of values a categorical variable can take. Then the Gram matrix K, containing the values of the kernel evaluated at each pair of points on the full grid, can be written as $K = \otimes_{d=1}^{D} K^{(d)}$, a Kronecker product of the Gram matrices $K^{(d)} \in \mathbb{R}^{n_d \times n_d}$ on each dimension [Saatçi, 2011]. The same holds for the Cholesky factor L where

$K = LL^\top$: we have $L = \otimes_{d=1}^{D} L^{(d)}$ where $K^{(d)} = L^{(d)} L^{(d)\top} \in \mathbb{R}^{n_d \times n_d}$. Then we define $\phi_d(x_i)$ to be the i^{th} row of $L^{(d)}$, so that $k_d(x_i, x_j) = K_{ij}^{(d)} = \phi_d(x_i)^\top \phi_d(x_j)$. In general a Cholesky decomposition for an m by m matrix takes $O(m^3)$ to compute. Thus $\phi_d(x_i)$ for $i = 1, \ldots, N$ require $O(n_d^3)$ to compute in total. Hence the computation of features become feasible even for large N as long as the n_d are reasonably small.

Random feature maps In most cases the data does not lie on a grid, nor can k_d be expressed as the inner product of finite feature vectors. In this case we can use random feature maps $\phi_d : \mathcal{X} \to \mathbb{R}^n$ where $\mathbb{E}[\phi_d(x)^\top \phi_d(x')] = k_d(x, x')$. An example is random Fourier features (RFF) [Rahimi and Recht, 2007] for stationary kernels, where $\mathbb{V}[\phi_d(x)^\top \phi_d(x')] = O(\frac{1}{n})$. So we are introducing a further approximation $k_d(x, x') \approx \phi_d(x)^\top \phi_d(x')$, with more accurate approximations for larger n. This is feasible even for large N as $\phi_d(x)$ only takes $O(n)$ computation. See Appendix 3.7.4 for details. For non-stationary kernels, we can obtain features by Nyström methods [Williams and Seeger, 2001, Drineas and Mahoney, 2005], which use a set of n inducing points to approximate K. The kernel is evaluated for each pair of inducing points and also between the inducing points and the data, giving matrices K_{nn} and K_{Nn}. Then $\hat{K} \approx K_{Nn} K_{nn}^{-1} K_{Nn}^\top = \Phi^\top \Phi$ where $\Phi = L_{nn}^{-1} K_{Nn}^\top$. Hence the columns of Φ can be defined to be the Nyström features.

3.7.6 Collaborative Filtering

Using Binary Vectors for Side Information

Note if the side information $\omega_1(u_i), \omega_2(v_j)$ are binary vectors with non-zeros at indices $\mathcal{I}_i, \mathcal{J}_j$ respectively, we have:

$$f(u_i, v_j) = (a_1 U_i + b_1 \sum_{k \in \mathcal{I}_i} U_{n_1+k})^T W (a_2 V_j + b_2 \sum_{k \in \mathcal{J}_j} V_{n_2+k})$$

which can be reparametrised to:

$$f(u_i, v_j) = a(U_i + b \sum_{k \in \mathcal{I}_i} U_{n_1+k})^T W (V_j + c \sum_{k \in \mathcal{J}_j} V_{n_2+k}) \tag{3.7}$$

Hyperparameter tuning for MovieLens 100K

Hyperparameters were tuned on the following values. For PMF and fixed W TGP: $\sigma_u = [0.3, 0.1, 0.03], \sigma^2 = [1.0, 0.1, 0.01, 0.001], \epsilon_u = [10^{-5}, 10^{-6}, 10^{-7}]$ where ϵ_u, ϵ_w are

the step sizes for SGD on U/V and W respectively. We noticed that for a fixed W the model overfits quickly in less than 30 epochs, whereas when learning W the test RMSE decreases steadily. So we used a different grid of parameters for tuning the models where W is learned: $\sigma_u = [0.3, 0.1], \sigma^2 = [1.0, 0.75], \epsilon_u, \epsilon_w = [10^{-5}, 10^{-6}]$. For models with side information, we tuned on $a = [0.25, 0.5, 0.75], b, c = [0.15, 0.3, 0.45]$.

3.7.7 California House Prices Data

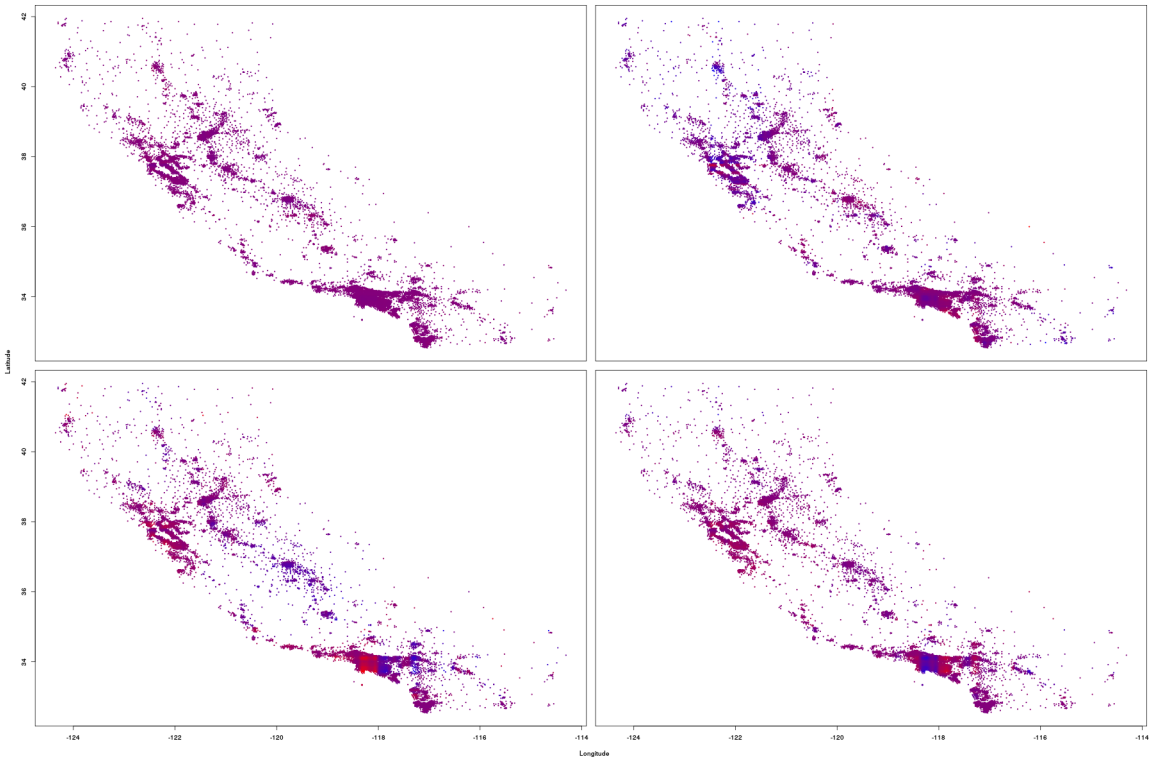


Figure 3.5: Heatmap showing the four additive components of predictions of the last sample of TGP for $r = 2, n = 200$, using uniform colouring scheme.

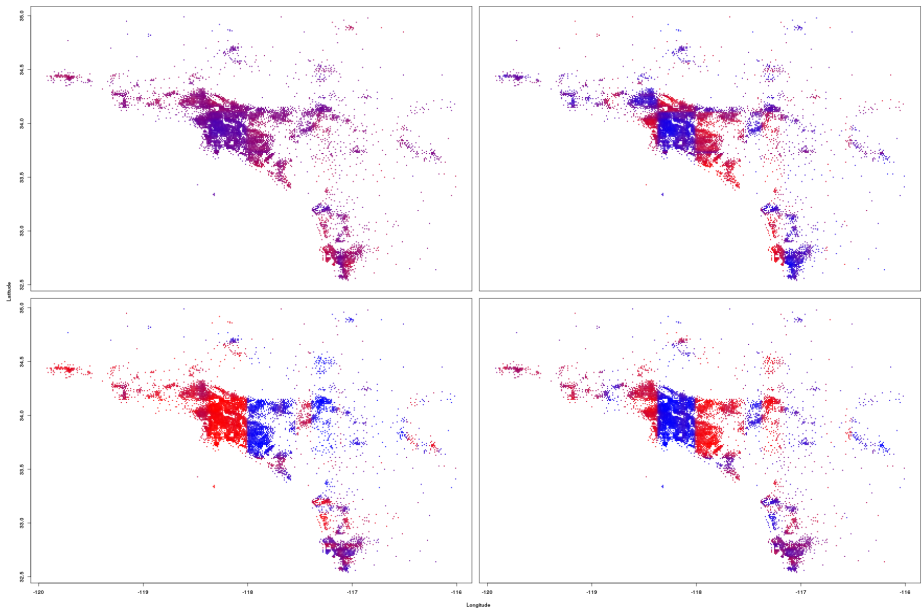


Figure 3.6: Zoom in on LA area of Figure 3.4.

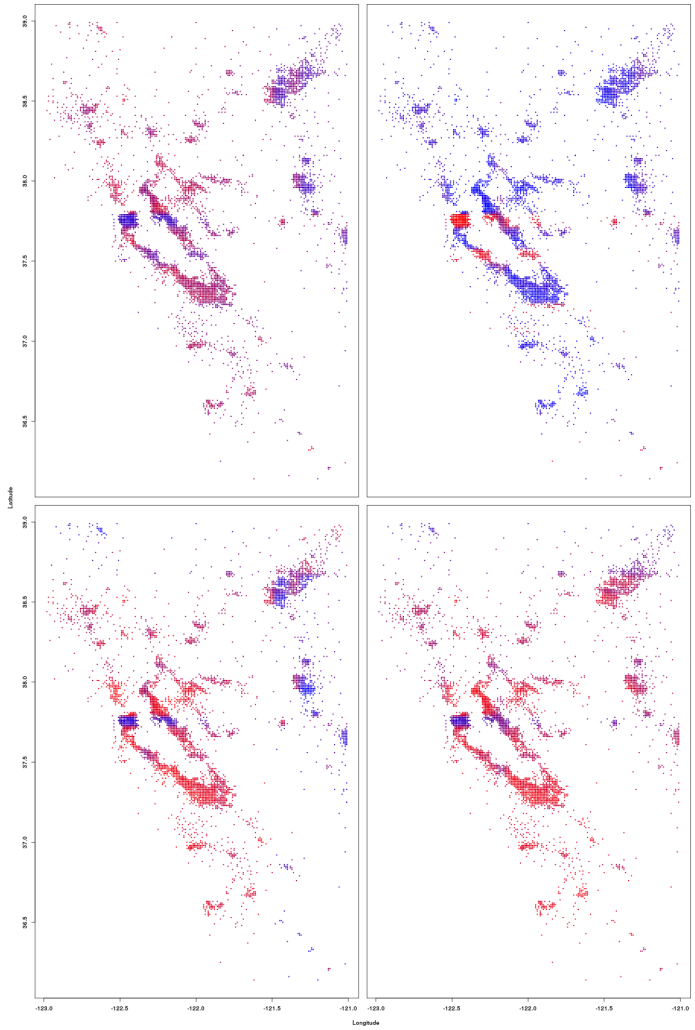


Figure 3.7: Zoom in on Bay area of Figure 3.4.

Table 3.2: Mean and standard deviation of Gelman Rubin statistic for HMC on TGP.

Model	$n = 25$	$n = 50$	$n = 100$	$n = 200$
TGP, $r = 2$	2.67 ± 1.34	2.55 ± 1.37	2.10 ± 0.70	1.92 ± 0.67
TGP, $r = 5$	1.06 ± 0.27	1.06 ± 0.19	1.15 ± 0.11	1.11 ± 0.11
TGP, $r = 10$	1.00 ± 0.03	1.02 ± 0.04	1.01 ± 0.02	1.06 ± 0.03

Table 3.3: Mean and standard deviation of Effective Sample Size (out of 1200) for HMC on TGP.

Model	$n = 25$	$n = 50$	$n = 100$	$n = 200$
TGP, $r = 2$	230 ± 459	5 ± 17	11 ± 81	12 ± 74
TGP, $r = 5$	244 ± 118	121 ± 70	34 ± 64	42 ± 79
TGP, $r = 10$	692 ± 152	196 ± 93	310 ± 111	96 ± 165

3.7.8 Irish Wind Data

Regression on spatio-temporal data with grid structure We use the Irish wind data [4] giving daily average wind speeds for 12 locations in Ireland between 1961 and 1978 (78,888 observations). We only use the covariates longitude, latitude and time. Note a 2D grid structure arises for the data when we treat the spatial covariates as one dimension and time as another. Again we whiten each covariate and observations, and use 20,000 randomly chosen data points for training and the rest for test. Using an isotropic SE kernel for space, and the sum of a periodic kernel and a SE kernel for time (to model annual periodicity and global trend), we first fit a GP efficiently exploiting the grid structure [Saatçi, 2011]. The optimised hyperparameters are then used to construct Cholesky features. Again we use NUTS for inference on both the *full-rank* model and TGP, using 4 chains with 100 warmup draws and 100 samples.

Table 3.4: Train/Test RMSE on Irish wind data.

Model	**Train RMSE**	**Test RMSE**
GP	4.8822	4.9915
Full-rank	4.8816	4.9898
TGP, $r = 2$	4.9120	4.9753
TGP, $r = 5$	4.8996	**4.9735**
TGP, $r = 10$	4.8913	4.9754

All models show good convergence after 100 warmup draws, indicated by the aforementioned convergence diagnostics. Looking at Table 3.4, we see similar patterns in the results for the wind data as for the house prices data: the GP, which is equivalent to the *full-rank* model with Cholesky features (confirmed by similar train/test RMSE), shows lower training error than TGP, whereas TGP shows superior predictive performance. These results again suggest that TGP is an effective regulariser towards simpler regression functions compared to GPs.

[4] Obtained from `http://www.inside-r.org/packages/cran/gstat/docs/wind`

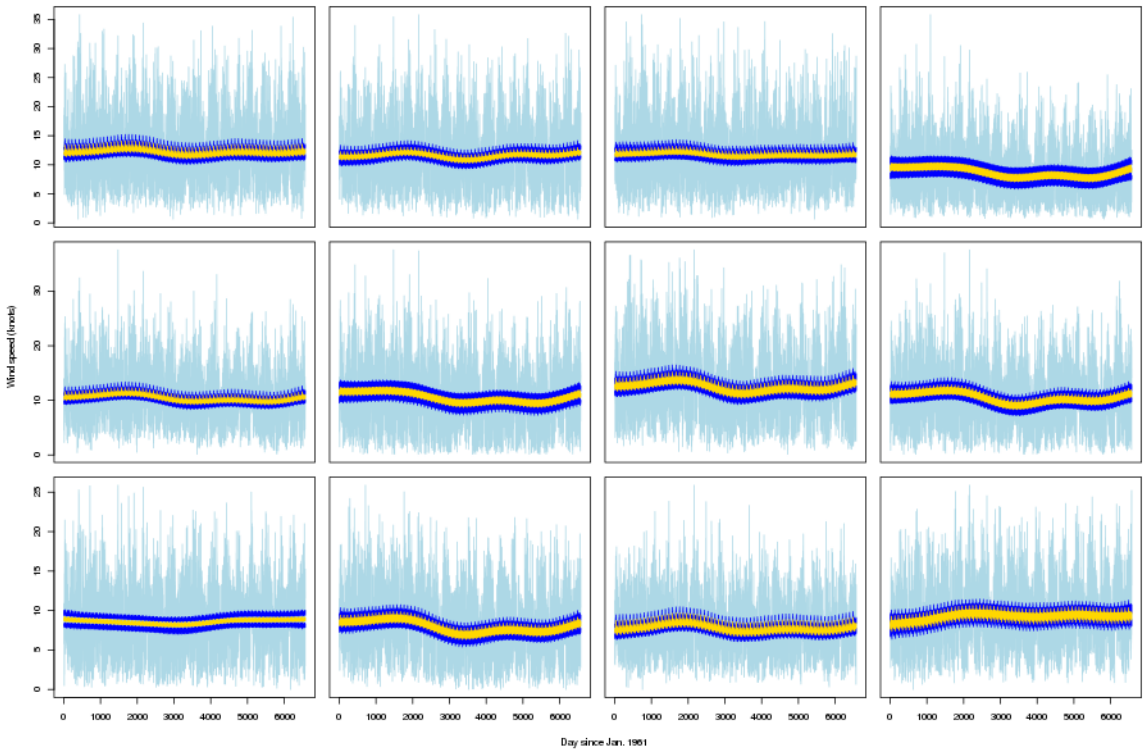


Figure 3.8: The predictions for TGP with $r = 5$ on the 12 locations. The light blue lines are the true observations, the yellow are the mean predictions, and the blue show 2.5% and 97.5% percentiles of predictions for samples.

3.7.9 Future Work

Note that TGP can easily be extended to non-Gaussian likelihoods, since all we need for SGD and HMC is the likelihood and priors to be analytic and differentiable in the parameters. For very high dimensions where even the r^D entries in W are undesirable, we can use a sparse representation of W with say Q non-zeros. All derivations carry forward, and we obtain time complexity $O(m(nrD + QD))$ for gradient computations in SGD. It would be interesting to compare TGP against other algorithms suitable for high-dimensional data. Furthermore, it would be desirable to have a sampling algorithm that scales sub-linearly, to benefit from the Bayesian approach to learning when N is large and HMC is infeasible. One example is Stochastic Gradient Langevin Dynamics (SGLD) [Welling and Teh, 2011] among many other Stochastic Gradient MCMC [Ma et al., 2015] algorithms. We have also tried mean-field variational inference, but results were poor compared to HMC. Moreover a more efficient method of tuning hyperparameters than by cross-validation would be ideal, especially for big N settings with many kernel hyperparameters. One potential solution is the fully Bayesian

approach, learning hyperparameters directly by imposing priors and sampling or MAP. We leave these extensions for future work.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Collaborative Filtering with Side Information: a Gaussian Process Perspective
Publication Status	□Published □ Accepted for Publication □Submitted for Publication x Unpublished and unsubmitted work written in a manuscript style
Publication Details	Kim, H., Lu, X., Flaxman, S. and Teh, Y.W., 2016, May. Collaborative Filtering with Side Information: a Gaussian Process Perspective. arXiv preprint arXiv:1605.07025.

Student Confirmation

Student Name:	Hyun Jik Kim	
Contribution to the Paper	- One of two leads of the project, with help from two advisors. - Carried out most of the collaborative filtering experiments, as well as all experiments for the California House Prices data - Derived MAP & HMC and their time complexities. - Proved the elementwise convergence of TGP to standard Gaussian - Did most of the paper writing, including the literature review of related work.	
Signature	Date	

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Yee Whye Teh, Professor of Statistical Machine Learning		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

Chapter 4

Scaling up the Automatic Statistician: Scalable Structure Discovery using Gaussian Processes

The following chapter is a self-contained paper:

> **Kim, H.** and Teh, Y.W., 2018, April. Scaling up the Automatic Statistician: Scalable Structure Discovery using Gaussian Processes. In Artificial Intelligence and Statistics (AISTATS).

There is a statement of authorship at the end of this chapter.

Abstract

Automating statistical modelling is a challenging problem in artificial intelligence. The Automatic Statistician takes a first step in this direction, by employing a kernel search algorithm with Gaussian Processes (GP) to provide interpretable statistical models for regression problems. However this does not scale due to its $O(N^3)$ running time for the model selection. We propose Scalable Kernel Composition (SKC), a scalable kernel search algorithm that extends the Automatic Statistician to bigger data sets. In doing so, we derive a cheap upper bound on the GP marginal likelihood that sandwiches the marginal likelihood with the variational lower bound. We show that the upper bound is significantly tighter than the lower bound and thus useful for model selection.

4.1 Introduction

Automated statistical modelling is an area of research in its early stages, yet it is becoming an increasingly important problem [Guyon et al., 2016]. As a growing number of disciplines use statistical analyses and models to help achieve their goals, the demand for statisticians, machine learning researchers and data scientists is at an all time high. Automated systems for statistical modelling serves to assist such human resources, if not as a best alternative where there is a shortage.

An example of a fruitful attempt at automated statistical modelling in nonparametric regression is Compositional Kernel Search (CKS) [Duvenaud et al., 2013], an algorithm that fits a Gaussian Process (GP) to the data and automatically chooses a suitable parametric form of the kernel. This leads to high predictive performance that matches kernels hand-selected by GP experts [Rasmussen and Williams, 2005]. There also exist other approaches that tackle this model selection problem by using a more flexible kernel [Bach et al., 2004, Oliva et al., 2016, Samo and Roberts, 2015, Wilson and Adams, 2013, Wilson et al., 2016]. However the distinctive feature of Duvenaud et al. [2013] is that the resulting models are interpretable; the kernels are constructed in such a way that we can use them to describe patterns in the data, and thus can be used for automated exploratory data analysis. Lloyd et al. [2014] exploit this to generate natural language analyses from these kernels, a procedure that they name Automatic Bayesian Covariance Discovery (ABCD). The Automatic Statistician[1] implements this to output a 10-15 page report when given data input.

However, a limitation of ABCD is that it does not scale; due to the $O(N^3)$ time for inference in GPs, the analysis is constrained to small data sets, specialising in one dimensional time series data. This is undesirable not only because the average size of data sets is growing fast, but also because there is potentially more information in bigger data, implying a greater need for more expressive models that can discover finer structure. This paper proposes Scalable Kernel Composition (SKC), a scalable extension of CKS, to push the boundaries of automated interpretable statistical modelling to bigger data. In summary, our work makes the following contributions:

- We propose the first scalable version of the Automatic Statistician that scales up to medium-sized data sets by reducing algorithmic complexity from $O(N^3)$ to $O(N^2)$ and enhancing parallelisability.

[1] See `http://www.automaticstatistician.com/index/` for example analyses.

- We derive a novel cheap upper bound on the GP marginal likelihood, that is used in SKC with the variational lower bound [Titsias, 2009] to sandwich the GP marginal likelihood.
- We show that our upper bound is significantly tighter than the lower bound, and plays an important role for model selection.

4.2 ABCD and CKS

The Compositional Kernel Search (CKS) algorithm [Duvenaud et al., 2013] builds on the idea that the sum and product of two positive definite kernels are also positive definite. Starting off with a set $\mathcal{B}$ of base kernels defined on $\mathbb{R} \times \mathbb{R}$, the algorithm searches through the space of zero-mean GPs with kernels that can be expressed in terms of sums and products of these base kernels. $\mathcal{B} = \{\text{SE,LIN,PER}\}$ is used, which correspond to the squared exponential, linear and periodic kernel respectively (see Appendix 4.6.3 for the exact form of these base kernels). Thus candidate kernels form an open-ended space of GP models, allowing for an expressive model. Such approaches for structure discovery have also appeared in Gardner et al. [2017], Grosse et al. [2012]. A greedy search is employed to explore this space, with each kernel scored by the Bayesian Information Criterion (BIC) [Schwarz et al., 1978] [2] after optimising the kernel hyperparameters by type II maximum likelihood (ML-II). See Appendix 4.6.2 for the algorithm in detail.

The resulting kernel can be simplified to be expressed as a sum of products of base kernels, which has the notable benefit of interpretability. In particular, note $f_1 \sim GP(0, k_1), f_2 \sim GP(0, k_2) \Rightarrow f_1 + f_2 \sim GP(0, k_1 + k_2)$ for independent f_1 and f_2. So a GP whose kernel is a sum of products of kernels can be interpreted as sums of GPs each with structure given by the product of kernels. Now each base kernel in a product modifies the model in a consistent way. For example, multiplication by SE converts global structure into local structure since $\text{SE}(x, x')$ decreases exponentially with $|x - x'|$, and multiplication by PER is equivalent to multiplication of the modeled function by a periodic function (see Lloyd et al. [2014] for detailed interpretations of different combinations). This observation is used in Automatic Bayesian Covariance Discovery (ABCD) [Lloyd et al., 2014], giving a natural language description of the resulting function modeled by the composite kernel. In summary ABCD consists

[2] BIC = log marginal likelihood with a model complexity penalty. We use a definition where higher BIC means better model fit. See Appendix 4.6.1.

of two algorithms: the compositional kernel search CKS, and the natural language translation of the kernel into a piece of exploratory data analysis.

4.3 Scaling up ABCD

ABCD provides a framework for a natural extension to big data settings, in that we only need to be able to scale up CKS, then the natural language description of models can be directly applied. The difficulty of this extension lies in the $O(N^3)$ time for evaluation of the GP marginal likelihood and its gradients with respect to the kernel hyperparameters.

A naïve approach is to subsample the data to reduce N, but then we may fail to capture global structure such as periodicities with long periods or omit a set of points displaying a certain local structure. We show failure cases of random subsampling in Section 4.4.3. Regarding more strategic subsampling, the possibility of a generic subsampling algorithm for GPs that is able to capture the aforementioned properties of the data is a challenging research problem in itself.

Alternatively it is tempting to use either an approximate marginal likelihood or the exact marginal likelihood of an approximate model as a proxy for the exact likelihood [Quinonero-Candela and Rasmussen, 2005, Seeger et al., 2003, Snelson and Ghahramani, 2005, Titsias, 2009], especially with modern GP approximations scaling to large data sets with millions of data points [Hensman et al., 2013, Wilson et al., 2015]. However such scalable GP methods are limited in interpretability as they often behave very differently to the full GP, lacking guarantees for the chosen kernel to faithfully reflect the actual structure in the data. In other words, the real challenge is to scale up the GP while preserving interpretability, and this is a difficult problem due to the tradeoff between scalability and accuracy of GP approximations. Our work pushes the frontiers of interpretable GPs to medium-sized data ($N = 10K \sim 100K$) by reducing the computational complexity of ABCD from $O(N^3)$ to $O(N^2)$. Extending this to large data sets ($N = 100K \sim 1M$) is a difficult open problem.

Our approach is as follows: we provide a cheap lower bound and upper bound to sandwich the exact marginal likelihood, and we use this interval for model selection. To do so we give a brief overview of the relevant work on low rank kernel approximations used for scaling up GPs, and we later outline how they can be applied to obtain cheap lower and upper bounds.

4.3.1 Nyström Methods and Sparse GPs

The Nyström Method [Drineas and Mahoney, 2005, Williams and Seeger, 2001] selects a set of m inducing points in the input space $\mathbb{R}^D$ that attempt to explain all the covariance in the Gram matrix of the kernel; the kernel is evaluated for each pair of inducing points and also between the inducing points and the data, giving matrices $K_{mm}, K_{mN} = K_{Nm}^\top$. This is used to create the Nyström approximation $\hat{K} = K_{Nm}(K_{mm})^\dagger K_{mN}$ of $K = K_{NN}$, where $\dagger$ is the pseudo-inverse. Applying Cholesky decomposition to K_{mm}, we see that the approximation admits the low-rank form $\Phi^\top\Phi$ and so allows efficient computation of determinants and inverses in $O(Nm^2)$ time (see Appendix 4.6.4). We later use the Nyström approximation to give an upper bound on the exact log marginal likelihood.

The Nyström approximation arises naturally in the sparse GP literature, where certain distributions are approximated by simpler ones involving $\boldsymbol{f_m}$, the GP evaluated at the m inducing points: the Deterministic Training Conditional (DTC) approximation [Seeger et al., 2003] defines a model that gives the marginal likelihood $q(\mathbf{y}) = \mathcal{N}(\mathbf{y}|0, \hat{K} + \sigma^2 I)$ ($\mathbf{y}$ is the vector of observations), whereas the Fully Independent Conditional (FIC) approximation [Snelson and Ghahramani, 2005] gives $q(\mathbf{y}) = \mathcal{N}(\mathbf{y}|0, \hat{K} + \mathrm{diag}(K - \hat{K}) + \sigma^2 I)$, correcting the Nyström approximation along the diagonals. The Partially Independent Conditional (PIC) approximation [Quinonero-Candela and Rasmussen, 2005] further improves this by correcting the Nyström approximation on block diagonals, with blocks typically of size $m \times m$. Note that the approximation is no longer low rank for FIC and PIC, but matrix inversion can still be computed in $O(Nm^2)$ time by Woodbury's Lemma (see Appendix 4.6.4).

The variational inducing points method (VAR) [Titsias, 2009] is rather different to DTC/FIC/PIC in that it gives the following variational lower bound on the exact log marginal likelihood:

$$\log[\mathcal{N}(\mathbf{y}|0, \hat{K} + \sigma^2 I)] - \frac{1}{2\sigma^2}\,\mathrm{Tr}(K - \hat{K}) \tag{4.1}$$

This lower bound is optimised with respect to the inducing points and the kernel hyperparameters, which is shown in the paper to successfully yield tight lower bounds in $O(Nm^2)$ time for reasonable values of m. Another useful property of VAR is that the lower bound can only increase as the set of inducing points grows [Matthews et al., 2016, Titsias, 2009]. It is also known that VAR always improves with extra computation, and that it successfully recovers the true posterior GP in most cases, contrary to other sparse GP methods [Bauer et al., 2016]. Hence this is what we

use in SKC to obtain a lower bound on the marginal likelihood and optimise the hyperparameters. We use 10 random initialisations of hyperparameters and choose the one with highest lower bound after optimisation.

4.3.2 A cheap and tight upper bound on the log marginal likelihood

Fixing the hyperparameters to be those tuned by VAR, we seek a cheap upper bound to the exact marginal likelihood. Upper bounds and lower bounds are qualitatively different, and in general it is more difficult to obtain an upper bound than a lower bound for the following reason: first note that the marginal likelihood is the integral of the likelihood with respect to the prior density of parameters. Hence to obtain a lower bound it suffices to exhibit regions in the parameter space giving high likelihood. However, to obtain an upper bound one must demonstrate the absence or lack of likelihood mass outside a certain region, an arguably more difficult task. There has been some work on the subject [Beal, 2003, Ji et al., 2010], but to the best of our knowledge there has not been any work on cheap upper bounds to the marginal likelihood in large N settings. So finding an upper bound from the perspective of the marginal likelihood can be difficult. Instead, we exploit the fact that the GP marginal likelihood has an analytic form, and treat it as a function of K. The GP log marginal likelihood is composed of two terms and a constant:

$$\begin{aligned}\log p(y) &= \log[\mathcal{N}(y|0, K+\sigma^2 I)] \\ &= -\frac{1}{2}\log\det(K+\sigma^2 I) \\ &\quad -\frac{1}{2}y^\top (K+\sigma^2 I)^{-1} y - \frac{N}{2}\log(2\pi)\end{aligned} \tag{4.2}$$

We give separate upper bounds on the negative log determinant (NLD) term and the negative inner product (NIP) term. For NLD, it has been proven that

$$-\frac{1}{2}\log\det(K+\sigma^2 I) \leq -\frac{1}{2}\log\det(\hat{K}+\sigma^2 I) \tag{4.3}$$

a consequence of $K-\hat{K}$ being a Schur complement of K and hence positive semi-definite (e.g. [Bardenet and Titsias, 2015]). So the Nyström approximation plugged into the NLD term serves as an upper bound that can be computed in $O(Nm^2)$ time (see Appendix 4.6.4).

As for NIP, we point out that $\lambda y^\top (K+\sigma^2 I)^{-1} y = \min_{f\in\mathcal{H}} \sum_{i=1}^N (y_i - f(x_i))^2 + \lambda \|f\|_{\mathcal{H}}^2$, the optimal value of the objective function in kernel ridge regression where $\mathcal{H}$ is the

Reproducing Kernel Hilbert Space associated with k (e.g. [Murphy, 2012b]). The dual problem, whose objective function has the same optimal value, is $\max_{\alpha \in \mathbb{R}^N} -\lambda[\alpha^\top (K + \sigma^2 I)\alpha - 2\alpha^\top y]$. So we have the following upper bound:

$$-\frac{1}{2} y^\top (K + \sigma^2 I)^{-1} y \leq \frac{1}{2} \alpha^\top (K + \sigma^2 I)\alpha - \alpha^\top y \tag{4.4}$$

$\forall \alpha \in \mathbb{R}^N$. Note that this is also in the form of an objective for conjugate gradients (CG) [Shewchuk et al., 1994], hence equality is obtained at the optimal value $\hat{\alpha} = (K + \sigma^2 I)^{-1} y$. We can approach the optimum for a tighter bound by using CG or preconditioned CG (PCG) until convergence or up to a fixed number of iterations to get a reasonable approximation to $\hat{\alpha}$. We used a maximum of $10m$ iterations but usually convergence is reached much faster, especially for higher values of m. Each iteration of CG and the computation of the upper bound takes $O(N^2)$ time, but PCG is very fast even for large data sets and using FIC/PIC as the preconditioner gives fastest convergence in general [Cutajar et al., 2016].

Recall that although the lower bound takes $O(Nm^2)$ to compute, we need $m = O(N^\beta)$ for accurate approximations, where β depends on the data distribution and kernel [Rudi et al., 2015]. β is usually close to 0.5, hence the lower bound is also effectively $O(N^2)$. In practice, the upper bound evaluation seems to be a little more expensive than the lower bound evaluation, but we only need to compute the upper bound once, whereas we must evaluate the lower bound and its gradients multiple times for the hyperparameter optimisation. We later confirm in Section 4.4.1 that the upper bound is fast to compute relative to the lower bound optimisation. We also show empirically that the upper bound is tighter than the lower bound in Section 4.4.1, and give the following sufficient condition for this to be true:

Proposition 2. *Suppose $(\hat{\lambda}_i)_{i=1}^N$ are the eigenvalues of $\hat{K} + \sigma^2 I$ in descending order. Then if (P)CG for the NIP term converges and $\hat{\lambda}_N \geq 2\sigma^2$, then the upper bound is tighter than the lower bound.*

Notice that $\hat{\lambda}_N \geq \sigma^2 \ \forall \hat{K}$, so the assumption is feasible. The proof is in Appendix 4.6.5.

We later show in Section 4.4 that the upper bound is not only tighter than the lower bound, but also much less sensitive to the choice of inducing points. Hence we use the upper bound to choose between kernels whose BIC intervals overlap.

Algorithm 1 Scalable Kernel Composition (SKC)

Input: data $x_1, \ldots, x_n \in \mathbb{R}^D, y_1, \ldots, y_n \in \mathbb{R}$, base kernel set $\mathcal{B}$, depth d, number of inducing points m, kernel buffer size S.

For each base kernel on each dimension, obtain lower and upper bounds to BIC (BIC interval), set k to be the kernel with highest upper bound, and add k to kernel buffer $\mathcal{K}$.
$\mathcal{C} \leftarrow \emptyset$
for depth= 1 **to** d **do**
  From $\mathcal{C}$, add to $\mathcal{K}$ all kernels whose intervals overlap with k if there are fewer than S of them, else add the kernels with top S upper bounds.
  for $k' \in \mathcal{K}$ **do**
    Add following kernels to $\mathcal{C}$ and obtain their BIC intervals:
      (1) All kernels of form $k' + B$ where B is any base kernel on any dimension
      (2) All kernels of form $k' \times B$ where B is any base kernel on any dimension
  end for
  if $\exists k^* \in \mathcal{C}$ with higher upper bound than k **then**
    $k \leftarrow k^*$
  end if
end for
Output: k, the resulting kernel.

4.3.3 SKC: Scalable Kernel Composition using the lower and upper bound

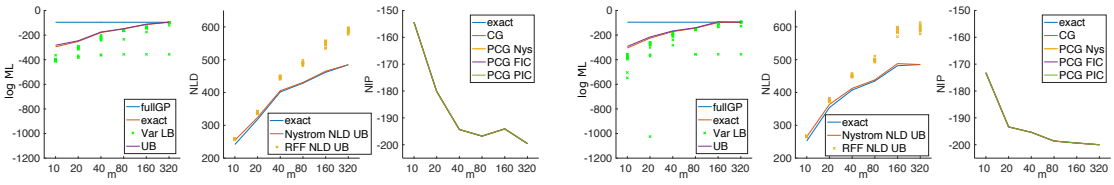
We base our algorithm on two intuitive claims. First, the lower and upper bounds converge to the true log marginal likelihood for fixed hyperparameters as the number of inducing points m increases. Second, the hyperparameters optimised by VAR converge to those obtained by optimising the exact log marginal likelihood as m increases. The former is confirmed in Figure 4.2 as well as in other works (e.g. [Bauer et al., 2016] for the lower bound), and the latter is confirmed in Appendix 4.6.6.

The algorithm proceeds as follows: for each base kernel and a fixed value of m, we compute the lower and upper bounds to obtain an interval for the GP marginal likelihood and hence the BIC of the kernel, with its hyperparameters optimised by VAR. We rank these kernels by their intervals, using the upper bound as a tie-breaker for kernels with overlapping intervals. We then perform a semi-greedy kernel search, expanding the search tree on all (or some, controlled by buffer size S) kernels whose intervals overlap with the top kernel at the current depth. We recurse to the next depth by computing intervals for these child kernels (parent kernel $+/\times$ base kernel), ranking them and further expanding the search tree. This is summarised in Algorithm

1, and Figure 4.12 in Appendix 4.6.17 is a visualisation of the tree for different values of m. Details on the optimisation and initialisation are given in Appendices 4.6.8 and 4.6.9.

The hyperparameters found by VAR may not be the global maximisers of the exact GP marginal likelihood, but as in ABCD we can optimise the marginal likelihood with multiple random seeds and choose the local optimum closest to the global optimum. One may still question whether the hyperparameter values found by VAR agree with the structure in the data, such as period values and slopes of linear trends. We show in Sections 4.4.2 and 4.4.3 that a small m suffices for this to be the case.

Choice of m We can guarantee that the lower bound increases with larger m, but cannot guarantee that the upper bound decreases, since we tend to get better hyperparameters for higher m that boost the marginal likelihood and hence the upper bound. We verify this in Section 4.4.1. Hence throughout SKC we fix m to be the largest possible value that one can afford, so that the marginal likelihood with hyperparameters optimised by VAR is as close as possible to the marginal likelihood with optimal hyperparameters. It is natural to wonder whether an adaptive choice of m is possible, using higher m for more promising kernels to tighten their bounds. However a fair comparison of different kernels via this lower and upper bound requires that they have the same value of m, since using a higher m is guaranteed to give a higher lower bound.



(a) Solar: fix inducing points

(b) Solar: learn inducing points

Figure 4.1: (a) Left: log marginal likelihood (ML) for fullGP with optimised hyperparameters, optimised VAR LB for each of 10 random initialisations per m, exact log ML for best hyperparameters out of 10, and corresponding UB. Middle: exact NLD and UB. Right: exact NIP and UB after m iterations of CG/PCG. (b) Same as Figure 4.1a, except learning inducing points for the LB optimisation and using them for subsequent computations.

Parallelisability Note that SKC is extremely parallelisable across different random initialisations and different kernels at each depth, as is CKS. In fact the presence of intervals for SKC and hence buffers of kernels allows further parallelisation over kernels of different depths in certain cases (see Appendix 4.6.7).

4.4 Experiments

4.4.1 Investigating the behaviour of the lower bound (LB) and upper bound (UB)

We present results for experiments showing the bounds we obtain for two small time series and a multidimensional regression data set, for which CKS is feasible. The first is the annual solar irradiance data from 1610 to 2011, with 402 observations [Lean et al., 1995]. The second is the time series Mauna Loa CO2 data [Tans and Keeling, 2005] with 689 observations. See Appendix 4.6.11 for plots of the time series. The multidimensional data set is the concrete compressive strength data set with 1030 observations and 8 covariates [Yeh, 1998]. The functional form of kernels used for each of these data sets have been found by CKS (see Figure 4.3). All observations and covariates have been normalised to have mean 0 and variance 1.

From the left of Figures 4.1a, 4.10a and 4.11a, (the latter two can be found in Appendix 4.6.17) we see that VAR gives a LB for the optimal log marginal likelihood that improves with increasing m. The best LB is tight relative to the exact log marginal likelihoods at the hyperparameters optimised by VAR. We also see that the UB is even tighter than the LB, and increases with m as hypothesised. From the middle plots, we observe that the Nyström approximation gives a very tight UB on the NLD term. We also tried using RFF to get a stochastic UB (see Appendix 4.6.16), but this is not as tight, especially for larger values of m. From the right plots, we can see that PCG with any of the three preconditioners (Nyström, FIC, PIC) give a very tight UB to the NIP term, whereas CG may require more iterations to get tight, for example in Figures 4.10a, 4.10b in Appendix 4.6.17.

Figure 4.2 shows further experimental evidence for a wider range of kernels reinforcing the claim that the UB is tighter than the LB, as well as being much less sensitive to the choice of inducing points. This is why the UB is a much more reliable metric for the model selection than the LB. Hence we use the UB for a one-off comparison of kernels when their BIC intervals overlap.

Comparing Figures 4.1a, 4.10a, 4.11a against 4.1b, 4.10b, 4.11b, learning inducing points does not lead to a vast improvement in the VAR LB. In fact the differences are not very significant, and sometimes learning inducing points can get the LB stuck in a bad local minimum, as indicated by the high variance of LB in the latter three figures. Moreover the differences in computational time is significant as we can see in

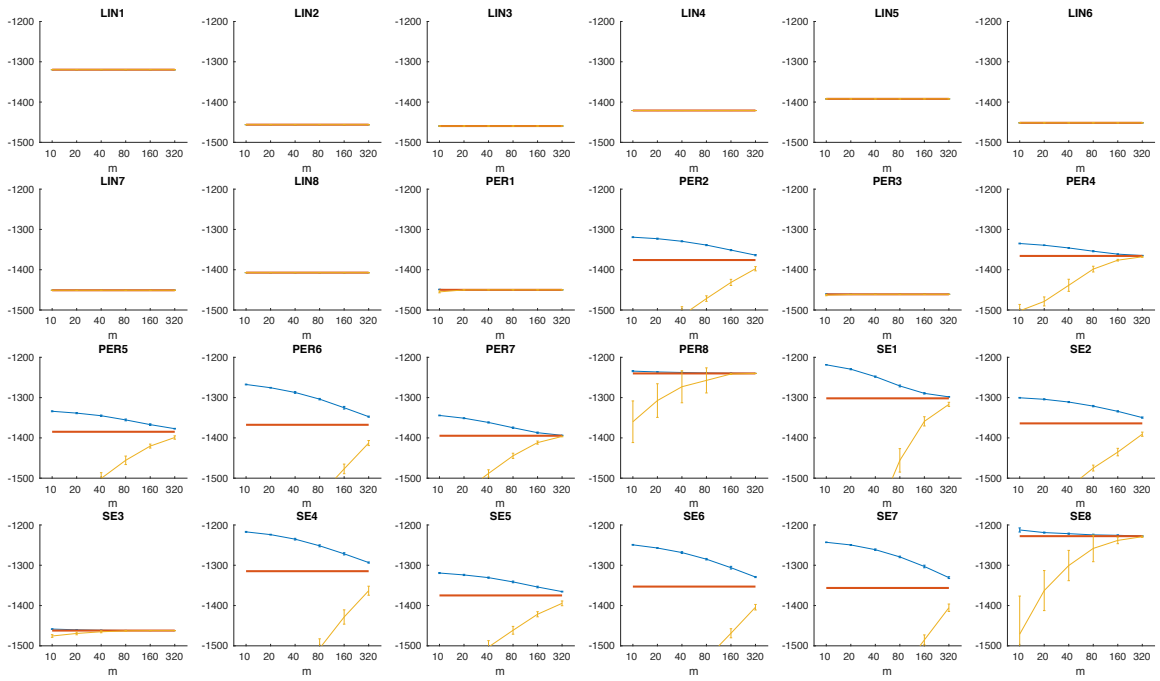


Figure 4.2: UB and LB for kernels at depth 1 on each dimension of concrete data, while varying the inducing points with hyperparameters fixed to the optimal values for the full GP. Error bars show mean ± 1 standard deviation over 10 random sets of inducing points.

Table 4.2 of Appendix 4.6.10. Hence the computation-accuracy trade-off is best when fixing the inducing points to a random subset of training data.

Table 4.2 also compares times for the different computations after fixing the inducing points. The gains from using the variational LB instead of the full GP is clear, especially for the larger data sets, and we also confirm that it is indeed the optimisation of the LB that is the bottleneck in terms of computational cost. We also see that the NIP UB computation times are similarly fast for all m, thus convergence of PCG with the PIC preconditioner is happening in only a few iterations.

4.4.2 SKC on small data sets

We compare the kernels chosen by CKS and by SKC for the three data sets. The results are summarised in Figure 4.3. For Solar, we see that SKC successfully finds SE × PER, which is the second highest kernel for CKS, with BIC very close to the top kernel. For Mauna, SKC selects (SE + PER)× SE + LIN, which is third highest for CKS and a BIC very close to the top kernel. Looking at the values of hyperparameters in kernels PER and LIN found by SKC, 40 inducing points are sufficient for it to

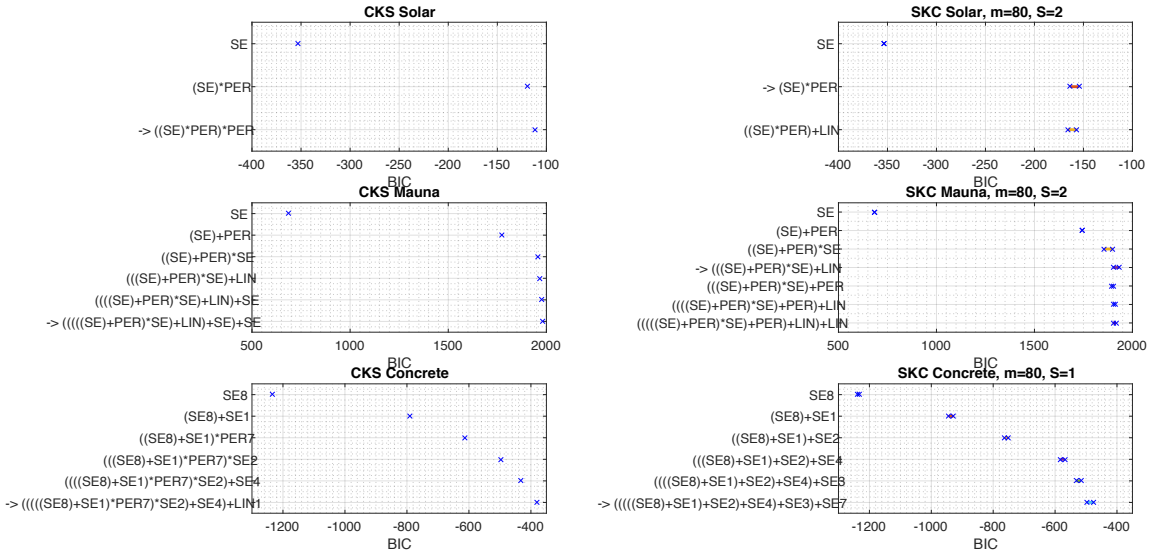
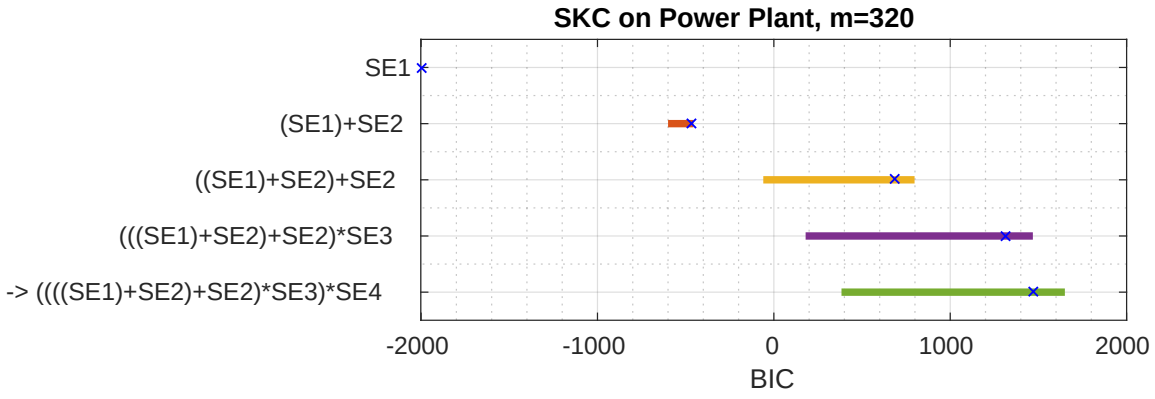


Figure 4.3: CKS & SKC results for up to depth 6. Left: BIC of kernels chosen at each depth by CKS. Right: BIC intervals of kernels that have been added to the buffer by SKC with $m = 80$. The arrow indicates the kernel chosen at the end.

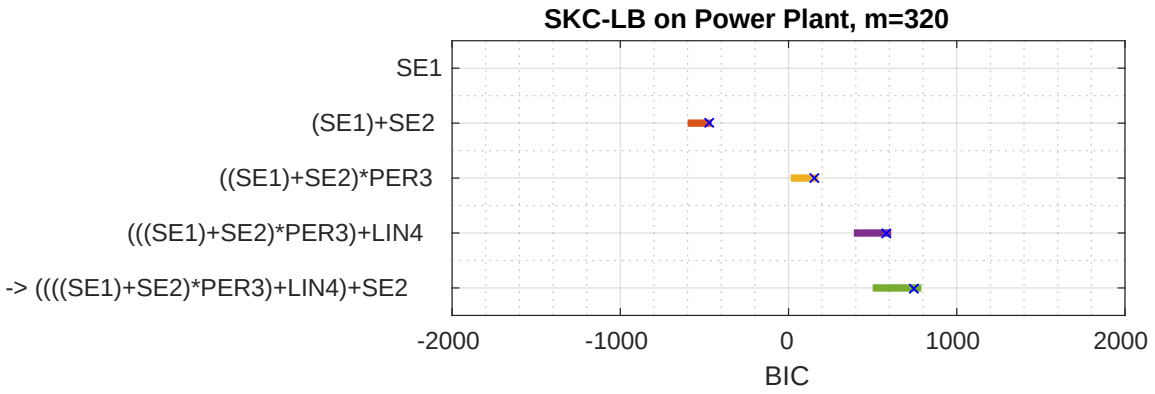
successfully find the correct periods and slopes in the data, reinforcing the claim that we only need a small m to find good hyperparameters (see Appendix 4.6.11 for details). For Concrete, a more challenging eight dimensional data set, we see that the kernels selected by SKC do not match those selected by CKS, but it still manages to find similar additive structure such as SE1+SE8 and SE4. Of course, the BIC intervals for kernels found by SKC are for hyperparameters found by VAR with $m = 80$, hence do not necessarily contain the optimal BIC of kernels in CKS. However the above results show that our method is still capable of selecting appropriate kernels even for low values of m, without having to home in to the optimal BIC using high values of m. The savings in computation time is significant even for these small data sets, as shown in Table 4.2.

4.4.3 SKC on medium-sized data sets & Why the lower bound is not enough

The UB has marginal benefits over the LB for small data sets, where the LB is already a fairly good estimate of the true BIC. However, this gap becomes significant as N grows and as kernels become more complex; the UB is much tighter and more stable with respect to the choice of inducing points (as shown in Figure 4.2), and plays a crucial role in the model selection.



(a) SKC



(b) SKC-LB

Figure 4.4: Comparison of (a) SKC and (b) SKC-LB, with $m = 320, S = 1$ to depth 5 on Power Plant data. The format is the same as Figure 4.3 but with the crosses at the true BIC instead of the bounds.

Power Plant We first show this for SKC on the Power Plant data with 9568 observations and 4 covariates [Tüfekci, 2014]. We see from Figure 4.4 that again the UB is much tighter than the LB, especially in more complex kernels further down the search tree. Comparing the two plots, we also see that the use of the UB in SKC has a significant positive impact on model selection: the kernel found by SKC at depth 5 has exact BIC 1469.6, whereas the corresponding kernel found by just using the LB (SKC-LB) has exact BIC 745.5. The pathological behaviour of SKC-LB occurs at depth 3, where the (SE1+SE2)*PER3 kernel found by SKC-LB has a higher LB than the corresponding SE1+SE2+SE2 kernel found by SKC. SKC-LB chooses the former kernel since it has a higher LB, which is a failure mode since the latter kernel has a significantly higher exact BIC. Due to the tightness of the UB, we see that SKC correctly chooses the latter kernel over the former, escaping the failure mode.

CKS vs SKC runtime We run CKS and SKC for $m = 160, 320, 640$ on Power Plant

data on the same machine with the same hyperparameter setting. The runtimes up to depths 1/2 are: 28.7h/94.7h (CKS), 0.6h/2.6h (SKC, $m = 160$), 1.1h/4.1h (SKC, $m = 320$), 4.2h/15.0h (SKC, $m = 640$). Again, the reduction in computation time for SKC are substantial.

Time Series We also implement SKC on medium-sized time series data to explore whether it can successfully find known structure at different scales. Such data sets occur frequently for hourly time series over several years or daily time series over centuries.

GEFCOM First we use the energy load data set from the 2012 Global Energy Forecasting Competition [Hong et al., 2014], and hourly time series of energy load from 2004/01/01 to 2008/06/30, with 8 weeks of missing data, giving $N = 38,070$. Notice that this data size is far beyond the scope of full GP optimisation in CKS.

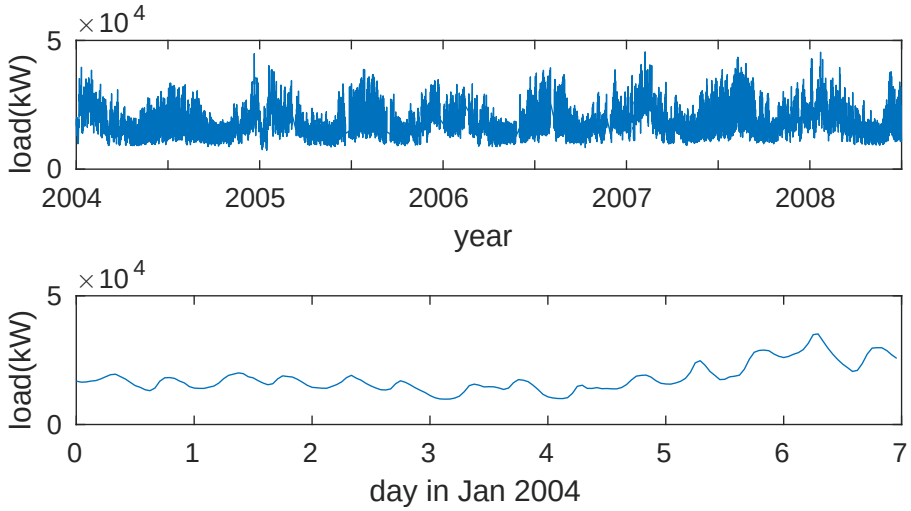


Figure 4.5: Top: plot of full GEFCOM data. Bottom: zoom in on the first 7 days.

From the plots, we can see that there is a noisy 6-month periodicity in the time series, as well as a clear daily periodicity with peaks in the morning and evening. Despite the periodicity, there are some noticeable irregularities in the daily pattern.

The $\mathrm{SE}_1 \times \mathrm{PER}_1 + \mathrm{SE}_2 \times (\mathrm{PER}_2 + \mathrm{LIN})$ kernel found by SKC with $m = 160$ and its hyperparameters are summarised in the first two columns of Table 4.1. Note that with only 160 inducing points, SKC has successfully found the daily periodicity (PER_1) and the 6-month periodicity (PER_2). The SE_1 kernel in the first additive term suggests a local periodicity, exhibiting the irregular observation pattern that is repeated daily. Also note that the hyperparameter corresponding to the longer periodicity is close but not exactly half a year, owing to the noisiness of the periodicity that is apparent in the

Table 4.1: Hyperparameters of kernels found by SKC on GEFCOM data and on Tidal data after normalising y. Length scales, periods, and location converted to original scale, σ^2 left as was found with normalised y.

GEFCOM		**Tidal**	
SE_1	$\sigma^2 = 0.44$ $l = 60.5$ days	SE	$\sigma^2 = 2.32$ $l = 82.5$ days
PER_1	$\sigma^2 = 1.10$ $l = 1089$ days $p = 1.003$ days	PER_1	$\sigma^2 = 5.17$ $l = 1026$ days $p = 0.538$ days
SE_2	$\sigma^2 = 0.18$ $l = 331$ days	LIN	$\sigma^2 = 0.18$ $loc =$ year 2015.0
PER_2	$\sigma^2 = 0.06$ $l = 170$ days $p = 174$ days	PER_2	$\sigma^2 = 0.08$ $l = 5974$ days $p = 0.500$ days
LIN	$\sigma^2 = 0.17$ $loc =$ year 2006.1	PER_3	$\sigma^2 = 0.21$ $l = 338$ days $p = 14.6$ days

data. Moreover the magnitude of the second additive term $SE_2 \times PER_2$ that contains the longer periodicity is 0.18×0.06, which is much smaller than the magnitude 0.44×1.10 of the first additive term $SE_1 \times PER_1$. This explicitly shows that the second term has less effect than the first term on the behaviour of the time series, hence the weaker long-term periodicity.

Running the kernel search just using the LB with the same hyperparameters as SKC, the algorithm is only able to detect the daily periodicity and misses the 6-month periodicity. This is consistent with the observation that the LB is loose compared to the UB, especially for bigger data sets, hence fails to evaluate the kernels correctly. We also tried selecting a random subset of the data (sizes $320, 640, 1280, 2560$) and running CKS. For even a subset of size 2560, the algorithm could not find the daily periodicity. None of the four subset size values were able to make it past depth 4 in the kernel search, struggling to find sophisticated structure.

Tidal We also run SKC on tidal data, namely the sea level measurements in Dover from 2013/01/03 to 2016/08/31 [BODC, 2017]. This is an hourly time series, with each observation taken to be the mean of four readings taken every 15 minutes, giving $N = 31,957$.

Looking at the bottom of Figure 4.6, we can find clear 12-hour periodicities and amplitudes that follow slightly noisy bi-weekly periodicities. The shorter periods, called semi-diurnal tides, are on average 12 hours 25 minutes ≈ 0.518 days long, and the bi-weekly periods arise due to gravitational effects of the moon [Morrissey et al., 1996].

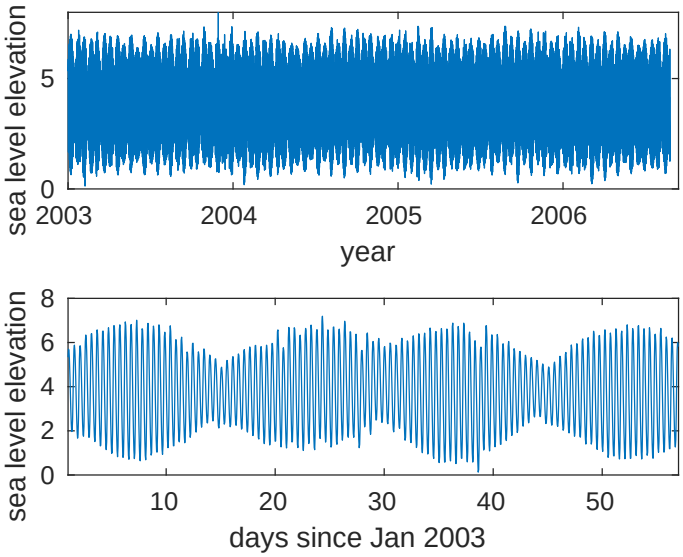


Figure 4.6: Top: plot of full tidal data. Bottom: zoom in on the first 4 weeks.

The SE $\times$ PER_1 $\times$ LIN $\times$ PER_2 $\times$ PER_3 kernel found by SKC with $m = 640$ and its hyperparameters are summarised in the right two columns of Table 4.1. The PER_1 $\times$ PER_3 kernel precisely corresponds to the doubly periodic structure in the data, whereby we have semi-daily periodicities with bi-weekly periodic amplitudes. The SE kernel has a length scale of 82.5 days, giving a local periodicity. This represents the irregularity of the amplitudes as can be seen on the bottom plot of Figure 4.6 between days 20 and 40. The LIN kernel has a large magnitude, but its effect is negligible when multiplied since the slope is calculated to be -5×10^{-6} (see Appendix 4.6.11 for the slope calculation formula). It essentially has the role of raising the magnitude of the resulting kernel and hence representing noise in the data. The PER_2 kernel is also negligible due to its high length scale and small magnitude, indicating that the amplitude of the periodicity due to this term is very small. Hence the term has minimal effect on the kernel.

The kernel search using just the LB fails to proceed past depth 3, since it is unable to find a kernel with a higher LB on the BIC than the previous depth (so the extra penalty incurred by increasing model complexity outweighs the increase in LB of log marginal likelihood). CKS on a random subset of the data similarly fails, the kernel search halting at depth 2 even for random subsets as large as size 2560.

In both cases, SKC is able to detect the structure of data and provide accurate numerical estimates of its features, all with much fewer inducing points than N, whereas the kernel search using just the LB or a random subset of the data both fail.

4.5 Conclusion and Discussion

We have introduced SKC, a scalable kernel discovery algorithm that extends CKS and hence ABCD to bigger data sets. We have also derived a novel cheap upper bound to the GP marginal likelihood that sandwiches the marginal likelihood with the variational lower bound [Titsias, 2009], and use this interval in SKC for selecting between different kernels. The reasons for using an upper bound instead of just the lower bound for model selection are as follows: the upper bound allows for a semi-greedy approach, allowing us to explore a wider range of kernels and compensates for the suboptimality coming from local optima in the hyperparameter optimisation. Should we wish to restrict the range of kernels explored for computational efficiency, the upper bound is tighter and more stable than the lower bound, hence we may use the upper bound as a reliable tie-breaker for kernels with overlapping intervals. Equipped with this upper bound, our method can pinpoint global/local periodicities and linear trends in time series with tens of thousands of data points, which are well beyond the reach of its predecessor CKS.

For future work we wish to make the algorithm even more scalable: for large data sets where quadratic runtime is infeasible, we can apply stochastic variational inference for GPs [Hensman et al., 2013] to optimise the lower bound, the bottleneck of SKC, using mini-batches of data. Finding an upper bound that is cheap and tight enough for model selection would pose a challenge. Also one drawback of the upper bound that could perhaps be resolved is that hyperparameter tuning by optimising the upper bound is difficult (see Appendix 4.6.12). One other minor scope for future work is using more accurate estimates of the model evidence than BIC. A related paper uses Laplace approximation instead of BIC for kernel evaluation in a similar kernel search context [Malkomes et al., 2016]. However Laplace approximation adds on an expensive Hessian term, for which it is unclear how one can obtain lower and upper bounds.

Acknowledgements

HK and YWTs research leading to these results has received funding from the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 617071. We would also like to thank Michalis Titsias for helpful discussions.

4.6 Supplementary Material

4.6.1 Bayesian Information Criterion (BIC)

The BIC is a model selection criterion that is the marginal likelihood with a model complexity penalty:

$$BIC = \log p(y|\hat{\theta}) - \frac{1}{2} p \log(N)$$

for observations y, number of observations N, maximum likelihood estimate (MLE) of model hyperparameters $\hat{\theta}$, number of hyperparameters p. It is derived as an approximation to the log model evidence $\log p(y)$.

4.6.2 Compositional Kernel Search Algorithm

Algorithm 2 Compositional Kernel Search Algorithm

Input: data $x_1, \ldots, x_n \in \mathbb{R}^D, y_1, \ldots, y_n \in \mathbb{R}$,base kernel set $\mathcal{B}$
For each base kernel on each dimension, fit GP to data (i.e. optimise hyperparams by ML-II) and set k to be kernel with highest BIC.
for depth= 1 **to** d (either fix d or repeat until BIC no longer increases) **do**
  Fit GP to following kernels and set k to be the one with highest BIC:
  (1) All kernels of form $k + B$ where B is any base kernel on any dimension
  (2) All kernels of form $k \times B$ where B is any base kernel on any dimension
  (3) All kernels where a base kernel in k is replaced by another base kernel
end for
Output: k, the resulting kernel

4.6.3 Base Kernels

$$\begin{aligned}
\mathrm{LIN}(x, x') &= \sigma^2 (x - l)(x' - l) \\
\mathrm{SE}(x, x') &= \sigma^2 \exp\Big(- \frac{(x - x')^2}{2l^2} \Big) \\
\mathrm{PER}(x, x') &= \sigma^2 \exp\Big(- \frac{2 \sin^2(\pi(x - x')/p)}{l^2} \Big)
\end{aligned}$$

4.6.4 Matrix Identities

Lemma 5 (Woodbury's Matrix Inversion Lemma)**.** $(A+UBV)^{-1} = A^{-1} - A^{-1}U(B^{-1} + VA^{-1}U)^{-1}VA^{-1}$

So setting $A = \sigma^2 I$ (Nyström) or $\sigma^2 I + diag(K - \hat{K})$ (FIC) or $\sigma^2 I + blockdiag(K - \hat{K})$ (PIC), $U = \Phi^T = V$, $B = I$, we get:

$$(A + \Phi^\top \Phi)^{-1} = A^{-1} - A^{-1}\Phi^\top (I + \Phi A^{-1} \Phi^\top)^{-1} \Phi A^{-1}$$

Lemma 6 (Sylvester's Determinant Theorem)**.** $\det(I + AB) = \det(I + BA)\ \forall A \in \mathbb{R}^{m \times n}\ \forall B \in \mathbb{R}^{n \times m}$

Hence:

$$\begin{aligned}
\det(\sigma^2 I + \Phi^\top \Phi) &= (\sigma^2)^n \det(I + \sigma^{-2} \Phi^\top \Phi) \\
&= (\sigma^2)^n \det(I + \sigma^{-2} \Phi \Phi^\top) \\
&= (\sigma^2)^{n-m} \det(\sigma^2 I + \Phi \Phi^\top)
\end{aligned}$$

4.6.5 Proof of Proposition 1

Proof. If PCG converges, the upper bound for NIP is exact. We showed in Section 4.4.1 that the convergence happened in only a few iterations. Moreover Cortes et al [Cortes et al., 2010] shows that the lower bound for NIP can be rather loose in general.

So it suffices to prove that the upper bound for NLD is tighter than the lower bound for NLD. Let $(\lambda_i)_{i=1}^N, (\hat{\lambda}_i)_{i=1}^N$ be the ordered eigenvalues of $K + \sigma^2 I, \hat{K} + \sigma^2 I$ respectively. Since $K - \hat{K}$ is positive semi-definite (e.g. [Bardenet and Titsias, 2015]), we have $\lambda_i \geq \hat{\lambda}_i \geq 2\sigma^2\ \forall i$ (using the assumption in the proposition). Now the slack in the upper bound is:

$$\begin{aligned}
&-\frac{1}{2}\log\det(\hat{K} + \sigma^2 I) - (-\frac{1}{2}\log\det(K + \sigma^2 I)) \\
&\quad = \frac{1}{2}\sum_{i=1}^{N}(\log\lambda_i - \log\hat{\lambda}_i)
\end{aligned}$$

Hence the slack in the lower bound is:

$$\begin{aligned}
&-\frac{1}{2}\log\det(K + \sigma^2 I) \\
&\quad - \Big[-\frac{1}{2}\log\det(\hat{K} + \sigma^2 I) - \frac{1}{2\sigma^2}\operatorname{Tr}(K - \hat{K}) \Big] \\
&\quad = -\frac{1}{2}\sum_{i=1}^{N}(\log\lambda_i - \log\hat{\lambda}_i) + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(\lambda_i - \hat{\lambda}_i)
\end{aligned}$$

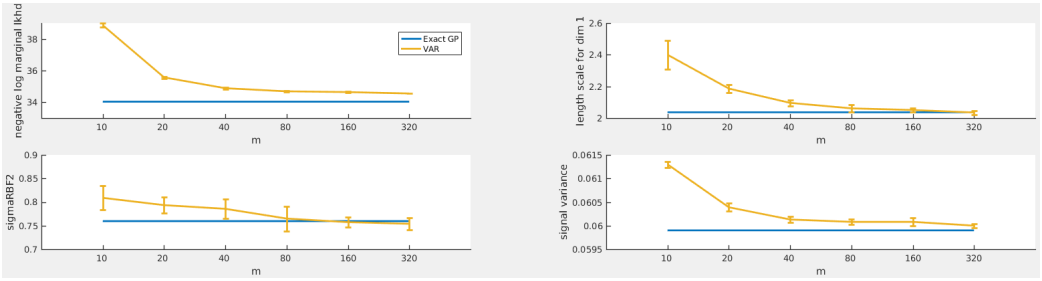


Figure 4.7: Log marginal likelihood and hyperparameter values after optimising the lower bound with ARD kernel on a subset of the Power plant data for different values of m. This is compared against the exact GP values when optimising the true log marginal likelihood. Error bars show mean $\pm$ 1 standard deviation over 10 random iterations.

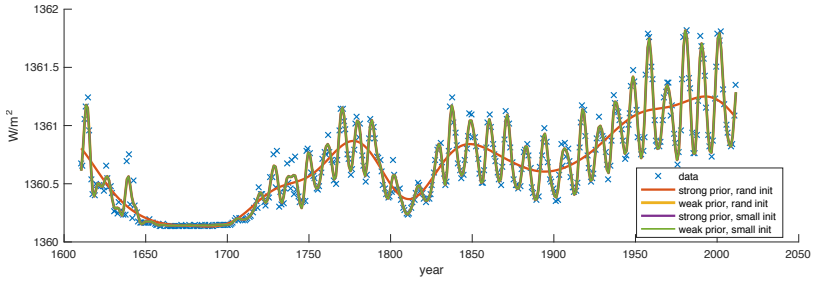


Figure 4.8: GP predictions on solar data set with SE kernel for different priors and initialisations.

Now by concavity and monotonicity of log, and since $\hat{\lambda} \geq 2\sigma^2$, we have:

$$\begin{aligned}
\frac{\log \lambda_i - \log \hat{\lambda}_i}{\lambda_i - \hat{\lambda}_i} &\leq \frac{1}{2\sigma^2} \\
\Rightarrow \sum_{i=1}^{N} (\log \lambda_i - \log \hat{\lambda}_i) &\leq \frac{1}{2\sigma^2} \sum_{i=1}^{N} (\lambda_i - \hat{\lambda}_i) \\
\Rightarrow \frac{1}{2} \sum_{i=1}^{N} (\log \lambda_i - \log \hat{\lambda}_i) &\leq \frac{1}{2\sigma^2} \sum_{i=1}^{N} (\lambda_i - \hat{\lambda}_i) - \frac{1}{2} \sum_{i=1}^{N} (\log \lambda_i - \log \hat{\lambda}_i)
\end{aligned}$$

□

4.6.6 Convergence of hyperparameters from optimising lower bound to optimal hyperparameters

Note from Figure 4.7 that the hyperparameters found by optimising the lower bound converges to the hyperparameters found by the exact GP when optimising the exact marginal likelihood, giving empirical evidence for the second claim in Section 4.3.3.

4.6.7 Parallelising SKC

Note that SKC can be parallelised across the random hyperparameter initialisations, and also across the kernels at each depth for computing the BIC intervals. In fact, SKC is even more parallelisable with the kernel buffer: say at a certain depth, we have two kernels remaining to be optimised and evaluated before we can move onto the next depth. If the buffer size is 5, say, then we can in fact move on to the next depth and grow the kernel search tree on the top 3 kernels of the buffer, without having to wait for the 2 kernel evaluations to be complete. This saves a lot of computation time wasted by idle cores waiting for all kernel evaluations to finish before moving on to the next depth of the kernel search tree.

4.6.8 Optimisation

Since we wish to use the learned kernels for interpretation, it is important to have the hyperparameters lie in a sensible region after the optimisation. In other words, we wish to regularise the hyperparameters during optimisation. For example, we want the SE kernel to learn a globally smooth function with local variation. When naïvely optimising the lower bound, sometimes the length scale and the signal variance becomes very small, so the SE kernel explains all the variation in the signal and ends up connecting the dots. We wish to avoid this type of behaviour. This can be achieved by giving priors to hyperparameters and optimising the energy (log prior added to the log marginal likelihood) instead, as well as using sensible initialisations. Looking at Figure 4.8, we see that using a strong prior with a sensible random initialisation (see Appendix 4.6.9 for details) gives a sensible smoothly varying function, whereas for all the three other cases, we have the length scale and signal variance shrinking to small values, causing the GP to overfit to the data. Note that the weak prior is the default prior used in the GPstuff software [Vanhatalo et al., 2013].

Careful initialisation of hyperparameters and inducing points is also very important, and can have strong influence the resulting optima. It is sensible to have the optimised hyperparameters of the parent kernel in the search tree be inherited and used to initialise the hyperparameters of the child. The new hyperparameters of the child must be initialised with random restarts, where the variance is small enough to ensure that they lie in a sensible region, but large enough to explore a good portion of this region. As for the inducing points, we want to spread them out to capture both local and global structure. Trying both K-means and a random subset of training data, we

conclude that they give similar results and resort to a random subset. Moreover we also have the option of learning the inducing points. However, this will be considerably more costly and show little improvement over fixing them, as we show in Section 4.4. Hence we do not learn the inducing points, but fix them to a given randomly chosen set.

Hence for SKC, we use *maximum a posteriori* (MAP) estimates instead of MLE for the hyperparameters to calculate the BIC, since the priors have a noticeable effect on the optimisation. This is justified [Murphy, 2012b] and has been used for example in Fraley and Raftery [2007], where they argue that using the MLE to estimate the BIC for Gaussian mixture models can fail due to singularities and degeneracies.

4.6.9 Hyperparameter initialisation and priors

$Z \sim \mathcal{N}(0,1), TN(\sigma^2, I)$ is a Gaussian with mean 0 and variance σ^2 truncated at the interval I then renormalised.

Signal noise

$\sigma^2 = 0.1 \times \exp(Z/2)$

$p(\log \sigma^2) = \mathcal{N}(0, 0.2)$

LIN

$\sigma^2 = \exp(V)$ where $V \sim TN(1, [-\infty, 0]), l = \exp(\frac{Z}{2})$

$p(\log \sigma^2) = logunif$

$p(\log l) = logunif$

SE

$l = \exp(Z/2), \sigma^2 = 0.1 \times \exp(Z/2)$

$p(\log l) = \mathcal{N}(0, 0.01), p(\log \sigma^2) = logunif$

PER

$p_{min} = \log(10 \times \frac{\max(x)-\min(x)}{N})$ (shortest possible period is 10 time steps)

$p_{max} = \log(\frac{\max(x)-\min(x)}{5})$ (longest possible period is a fifth of the range of data set)

$l = \exp(Z/2), p = \exp(p_{min} + W)$ or $\exp(p_{max} + U)$, $\sigma^2 = 0.1 \times \exp(Z/2)$ w.p. $\frac{1}{2}$

where $W \sim \mathcal{TN}(-0.5, [0, \infty])$,

$U \sim \mathcal{TN}(-0.5, [-\infty, 0])$)

$p(\log l) = t(\mu = 0, \sigma^2 = 1, \nu = 4)$,

$p(\log p) = \mathcal{LN}(p_{min} - 0.5, 0.25)$ or $\mathcal{LN}(p_{max} - 2, 0.5)$ w.p. $\frac{1}{2}$

Table 4.2: Mean and standard deviation of computation times (in seconds) for full GP optimisation, Var GP optimisation with and without learning inducing points (that involves many computations of the lower bound and gradeints), single NLD and NIP (PCG using PIC preconditioner) upper bound computation over 10 random iterations.

		Solar	**Mauna**	**Concrete**
GP		29.1950 ± 5.1430	164.8828 ± 58.7865	403.8233 ± 127.0364
Var GP	m=10	7.0259 ± 4.3928	6.0117 ± 3.8267	5.4358 ± 0.7298
	m=20	8.3121 ± 5.4763	11.9245 ± 6.8790	10.2410 ± 2.5109
	m=40	10.2263 ± 4.1025	17.1479 ± 10.7898	19.6678 ± 4.3924
	m=80	9.6752 ± 6.5343	28.9876 ± 13.0031	47.2225 ± 13.1955
	m=160	25.6330 ± 8.7934	91.0406 ± 39.8409	158.9199 ± 18.1276
	m=320	76.3447 ± 20.3337	202.2369 ± 96.0749	541.4835 ± 99.6145
NLD	m=10	0.0019 ± 0.0001	0.0033 ± 0.0002	0.0113 ± 0.0004
	m=20	0.0026 ± 0.0001	0.0046 ± 0.0002	0.0166 ± 0.0007
	m=40	0.0043 ± 0.0001	0.0079 ± 0.0003	0.0286 ± 0.0005
	m=80	0.0084 ± 0.0002	0.0154 ± 0.0004	0.0554 ± 0.0012
	m=160	0.0188 ± 0.0006	0.0338 ± 0.0007	0.1188 ± 0.0030
	m=320	0.0464 ± 0.0032	0.0789 ± 0.0036	0.2550 ± 0.0074
NIP	m=10	0.0474 ± 0.0092	0.1020 ± 0.0296	0.2342 ± 0.0206
	m=20	0.0422 ± 0.0130	0.1274 ± 0.0674	0.1746 ± 0.0450
	m=40	0.0284 ± 0.0075	0.0846 ± 0.0430	0.2345 ± 0.0483
	m=80	0.0199 ± 0.0081	0.0553 ± 0.0250	0.2176 ± 0.0376
	m=160	0.0206 ± 0.0053	0.0432 ± 0.0109	0.2136 ± 0.0422
	m=320	0.0250 ± 0.0019	0.0676 ± 0.0668	0.2295 ± 0.0433
Var GP, learn IP	m=10	23.4 ± 14.6	42.0 ± 33.0	110.0 ± 302.5
	m=20	38.5 ± 17.5	62.0 ± 66.0	70.0 ± 97.0
	m=40	124.7 ± 99.0	320.0 ± 236.0	307.0 ± 341.4
	m=80	268.6 ± 196.6	1935.0 ± 1103.0	666.0 ± 41.0
	m=160	1483.6 ± 773.8	10480.0 ± 5991.0	4786.0 ± 406.9
	m=320	2923.8 ± 1573.5	39789.0 ± 23870.0	25906.0 ± 820.9

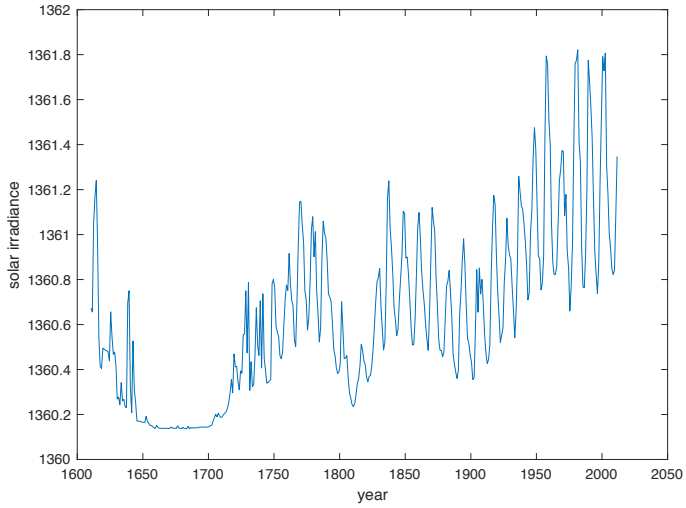
$p(\log \sigma^2) = logunif$ where $\mathcal{LN}(\mu, \sigma^2)$ is log Gaussian, $t(\mu, \sigma^2, \nu)$ is the student's t-distribution.

4.6.10 Computation times

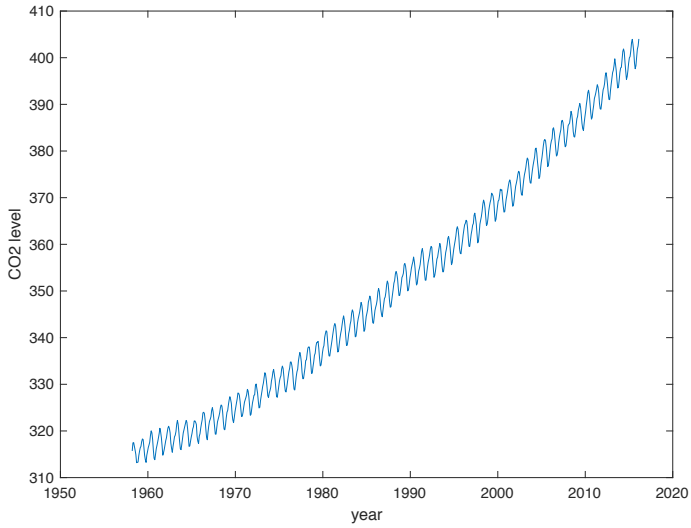
Look at Table 4.2. For all three data sets, the GP optimisation time is much greater than the sum of the Var GP optimisation time and the upper bound (NLD + NIP) evaluation time for $m \leq 80$. Hence the savings in computation time for SKC is significant even for these small data sets.

Note that we show the lower bound optimisation time against the upper bound evaluation time instead of the evaluation times for both, since this is what happens in SKC - the lower bound has to be optimised for each kernel, whereas the upper bound only has to be evaluated once.

4.6.11 Mauna and Solar plots and hyperparameter values found by SKC



(a) Solar



(b) Mauna

Figure 4.9: Plots of small time series data: Solar and Mauna

Solar The solar data has 26 cycles over 285 years, which gives a periodicity of around 10.9615 years. Using SKC with $m = 40$, we find the kernel: SE $\times$ PER $\times$ SE $\equiv$ SE $\times$ PER. The value of the period hyperparameter in PER is 10.9569 years, hence SKC finds the periodicity to 3 s.f. with only 40 inducing points. The SE term converts the global periodicity to local periodicity, with the extent of the locality governed by the length scale parameter in SE, equal to 45. This is fairly large, but smaller

than the range of the domain (1610-2011), indicating that the periodicity spans over a long time but isn't quite global. This is most likely due to the static solar irradiance between the years 1650-1700, adding a bit of noise to the periodicities.

Mauna The annual periodicity in the data and the linear trend with positive slope is clear. Linear regression gives us a slope of 1.5149. SKC with $m = 40$ gives the kernel: SE + PER + LIN. The period hyperparameter in PER takes value 1, hence SKC successfully finds the right periodicity. The offset l and magnitude σ^2 parameters of LIN allow us to calculate the slope by the formula $\sigma^2(x-l)^\top[\sigma^2(x-l)(x-l)^\top+\sigma_n^2 I]^{-1}y$ where σ_n^2 is the noise variance in the learned likelihood. This formula is obtained from the posterior mean of the GP, which is linear in the inputs for the linear kernel. This value amounts to 1.5150, hence the slope found by SKC is accurate to 3 s.f.

4.6.12 Optimising the upper bound

If the upper bound is tighter and more robust with respect to choice of inducing points, why don't we optimise the upper bound to find hyperparameters? If this were to be possible then we can maximise this to get an upper bound of the exact marginal likelihood with optimal hyperparameters. In fact this holds for any analytic upper bound whose value and gradients can be evaluated cheaply. Hence for any m, we can find an interval that contains the true optimised marginal likelihood. So if this interval is dominated by an interval of another kernel, we can discard the kernel and there is no need to evaluate the bounds for bigger values of m. Now we wish to use values of m such that we can choose the right kernel (or kernels) at each depth of the search tree with minimal computation. This gives rise to an exploitation-exploration trade-off, whereby we want to balance between raising m for tight intervals that allow us to discard unsuitable kernels whose intervals fall strictly below that of other kernels, and quickly moving on to the next depth in the search tree to search for finer structure in the data. The search algorithm is highly parallelisable, and thus we may raise m simultaneously for all candidate kernels. At deeper levels of the search tree, there may be too many candidates for simultaneous computation, in which case we may select the ones with the highest upper bound to get tighter intervals. Such attempts are listed below.

Note the two inequalities for the NLD and NIP terms:

$$\begin{aligned}
-\frac{1}{2}\log\det(\hat{K}+\sigma^2 I) - \frac{1}{2\sigma^2}\operatorname{Tr}(K-\hat{K}) \\
&\leq -\frac{1}{2}\log\det(K+\sigma^2 I) \\
&\leq -\frac{1}{2}\log\det(\hat{K}+\sigma^2 I) \qquad (4.5) \\
-\frac{1}{2}y^\top(\hat{K}+\sigma^2 I)^{-1}y \\
&\leq -\frac{1}{2}y^\top(K+\sigma^2 I)^{-1}y \\
&\leq -\frac{1}{2}y^\top(K+(\sigma^2+\operatorname{Tr}(K-\hat{K}))I)^{-1}y \qquad (4.6)
\end{aligned}$$

Where the first two inequalities come from Bardenet and Titsias [2015], the third inequality is a direct consequence of $K-\hat{K}$ being postive semi-definite, and the last inequality is from Michalis Titsias' lecture slides [3].

Also from 4.4, we have that

$$-\frac{1}{2}\log\det(\hat{K}+\sigma^2 I) + \frac{1}{2}\alpha^\top(K+\sigma^2 I)\alpha - \alpha^\top y$$

is an upper bound $\forall\alpha \in \mathbb{R}^N$. Thus one idea of obtaining a cheap upper bound to the optimised marginal likelihood was to solve the following maximin optimisation problem:

$$\max_{\theta}\min_{\alpha\in\mathbb{R}^N} -\frac{1}{2}\log\det(\hat{K}+\sigma^2 I) + \frac{1}{2}\alpha^\top(K+\sigma^2 I)\alpha - \alpha^\top y$$

One way to solve this cheaply would be by coordinate descent, where one maximises with respect to θ fixing α, then minimises with respect to α fixing θ. However σ tends to blow up in practice. This is because the expression is $O(-\log\sigma^2+\sigma^2)$ for fixed α, hence maximising with respect to σ pushes it towards infinity.

An alternative is to sum the two upper bounds above to get the upper bound

$$-\frac{1}{2}\log\det(\hat{K}+\sigma^2 I) - \frac{1}{2}y^\top(K+(\sigma^2+\operatorname{Tr}(K-\hat{K}))I)^{-1}y$$

However we found that maximising this bound gives quite a loose upper bound unless $m = O(N)$. Hence this upper bound is not very useful.

[3] http://www.aueb.gr/users/mtitsias/papers/titsiasNipsVar14.pdf

4.6.13 Random Fourier Features

Random Fourier Features (RFF) (a.k.a. Random Kitchen Sinks) was introduced by Rahimi and Recht [2007] as a low rank approximation to the kernel matrix. It uses the following theorem

Theorem 4 (Bochner's Theorem [Rudin, 1964])**.** *A stationary kernel k(d) is positive definite if and only if k(d) is the Fourier transform of a non-negative measure.*

to give an unbiased low-rank approximation to the Gram matrix $K = \mathbb{E}[\Phi^\top \Phi]$ with $\Phi \in \mathbb{R}^{m \times N}$. A bigger m lowers the variance of the estimate. Using this approximation, one can compute determinants and inverses in $O(Nm^2)$ time. In the context of kernel composition in 4.2, RFFs have the nice property that samples from the spectral density of the sum or product of kernels can easily be obtained as sums or mixtures of samples of the individual kernels (see Appendix 4.6.14). We use this later to give a memory-efficient upper bound on the exact log marginal likelihood in Appendix 4.6.16.

4.6.14 Random Features for Sums and Products of Kernels

For RFF the kernel can be approximated by the inner product of random features given by samples from its spectral density, in a Monte Carlo approximation, as follows:

$$\begin{aligned} k(x-y) = \int_{\mathbb{R}^D} e^{iv^\top (x-y)} d\mathbb{P}(v) &\propto \int_{\mathbb{R}^D} p(v) e^{iv^\top (x-y)} dv \\ &= \mathbb{E}_{p(v)}[e^{iv^\top x}(e^{iv^\top y})^*] \\ &= \mathbb{E}_{p(v)}[Re(e^{iv^\top x}(e^{iv^\top y})^*)] \\ &\approx \frac{1}{m}\sum_{k=1}^{m} Re(e^{iv_k{}^\top x}(e^{iv_k{}^\top y})^*) \\ &= \mathbb{E}_{b,v}[\phi(x)^\top \phi(y)] \end{aligned}$$

where $\phi(x) = \sqrt{\frac{2}{m}}(cos(v_1{}^\top x + b_1), \ldots, cos(v_m{}^\top x + b_m))$ with spectral frequencies v_k iid samples from $p(v)$ and b_k iid samples from $U[0, 2\pi]$.

Let k_1, k_2 be two stationary kernels, with respective spectral densities p_1, p_2 so that $k_1(d) = a_1\hat{p_1}(d), k_2(d) = a_2\hat{p_2}(d)$, where $\hat{p}(d) := \int_{\mathbb{R}^D} p(v)e^{iv^\top d}dv$. We use this convention as the Fourier transform. Note $a_i = k_i(0)$.

$$\begin{aligned} (k_1 + k_2)(d) &= a_1 \int p_1(v) e^{iv^\top d} dv + a_2 \int p_2(v) e^{iv^\top d} dv \\ &= (a_1 + a_2)\hat{p_+}(d) \end{aligned}$$

where $p_+(v) = \frac{a_1}{a_1+a_2}p_1(v) + \frac{a_2}{a_1+a_2}p_2(v)$, a mixture of p_1 and p_2. So to generate RFF for $k_1 + k_2$, generate $v \sim p_+$ by generating $v \sim p_1$ w.p. $\frac{a_1}{a_1+a_2}$ and $v \sim p_2$ w.p. $\frac{a_2}{a_1+a_2}$
Now for the product, suppose

$$(k_1 \cdot k_2)(d) = a_1 a_2 \hat{p_1}(d)\hat{p_2}(d) = a_1 a_2 \hat{p_*}(d)$$

Then $p_*(d)$ is the inverse fourier transform of $\hat{p_1}\hat{p_2}$, which is the convolution $p_1 * p_2(d) := \int_{\mathbb{R}^D} p_1(z)p_2(d-z)dz$. So to generate RFF for $k_1 \cdot k_2$, generate $v \sim p_*$ by generating $v_1 \sim p_1, v_2 \sim p_2$ and setting $v = v_1 + v_2$.
This is not applicable for non-stationary kernels, such as the linear kernel. We deal with this problem as follows:

Suppose ϕ_1, ϕ_2 are random features such that
$k_1(x, x') = \phi_1(x)^\top \phi_1(x'), \phi_2(x)^\top \phi_2(x'), \phi_i : \mathbb{R}^D \to \mathbb{R}^m$.
It is straightforward to verify that

$$(k_1 + k_2)(x, x') = \phi_+(x)^\top \phi_+(x')$$
$$(k_1 \cdot k_2)(x, x') = \phi_*(x)^\top \phi_*(x')$$

where $\phi_+(\cdot) = (\phi_1(\cdot)^\top, \phi_2(\cdot)^\top)^\top$ and $\phi_*(\cdot) = \phi_1(\cdot) \otimes \phi_2(\cdot)$. However we do not want the number of features to grow as we add or multiply kernels, since it will grow exponentially. We want to keep it to be m features. So we subsample m entries from ϕ_+ (or ϕ_*) and scale by factor $\sqrt{2}$ ($\sqrt{m}$ for ϕ_*), which will still give us unbiased estimates of the kernel provided that each term of the inner product $\phi_+(x)^\top \phi_+(x')$ (or $\phi_*(x)^\top \phi_*(x')$) is an unbiased estimate of $(k_1 + k_2)(x, x')$(or $(k_1 \cdot k_2)(x, x')$).
This is how we generate random features for linear kernels combined with other stationary kernels, using the features $\phi(x) = \frac{\sigma}{\sqrt{m}}(x, \ldots, x)^\top$.

4.6.15 Spectral Density for PER

From Solin and Särkkä [2014], we have that the spectral density of the PER kernel is:

$$\sum_{n=-\infty}^{\infty} \frac{I_n(l^{-2})}{\exp(l^{-2})} \delta\left(v - \frac{2\pi n}{p}\right)$$

where I is the modified Bessel function of the first kind.

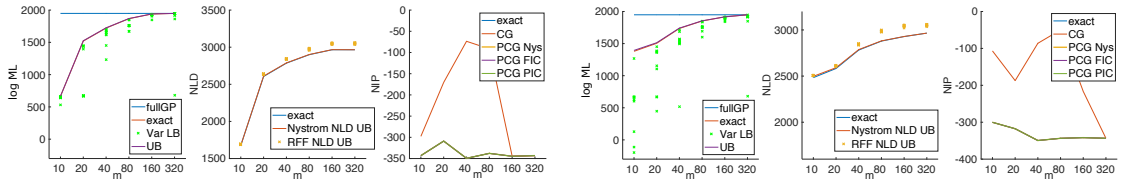
4.6.16 An upper bound to NLD using Random Fourier Features

Note that the function $f(X) = -\log\det(X)$ is convex on the set of positive definite matrices [Boyd and Vandenberghe, 2004]. Hence by Jensen's inequality we have, for $\Phi^\top\Phi$ an unbiased estimate of K:

$$\begin{aligned} -\frac{1}{2}\log\det(K+\sigma^2 I) &= f(K+\sigma^2 I) \\ &= f(\mathbb{E}[\Phi^\top\Phi + \sigma^2 I]) \\ &\leq \mathbb{E}[f(\Phi^\top\Phi + \sigma^2 I)] \end{aligned}$$

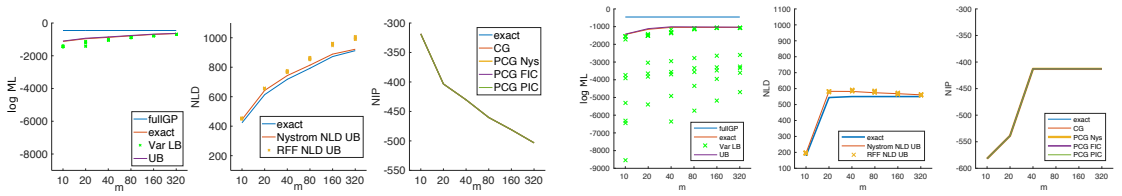
Hence $-\frac{1}{2}\log\det(\Phi^\top\Phi+\sigma^2 I)$ is a stochastic upper bound to NLD that can be calculated in $O(Nm^2)$. An example of such an unbiased estimator Φ is given by RFF.

4.6.17 Further Plots



(a) Mauna: fix inducing points (b) Mauna: learn inducing points

Figure 4.10: Same as 4.1a and 4.1b but for Mauna Loa data.



(a) Concrete: fix inducing points (b) Concrete: learn inducing points

Figure 4.11: Same as 4.1a and 4.1b but for Concrete data.

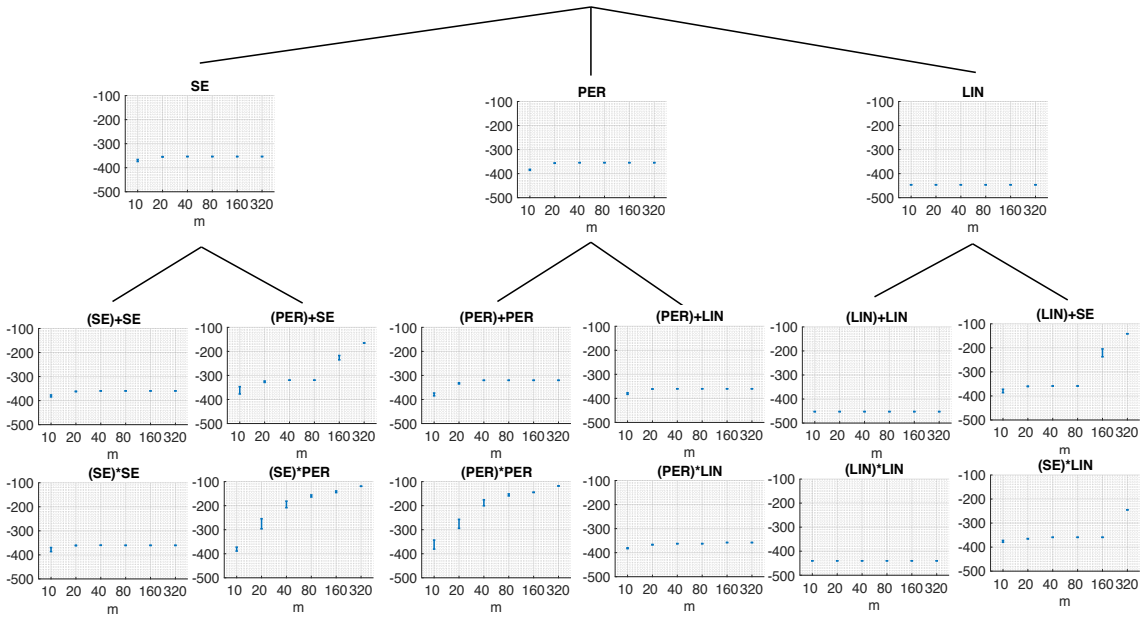


Figure 4.12: Kernel search tree for SKC on solar data up to depth 2. We show the upper and lower bounds for different numbers of inducing points m.

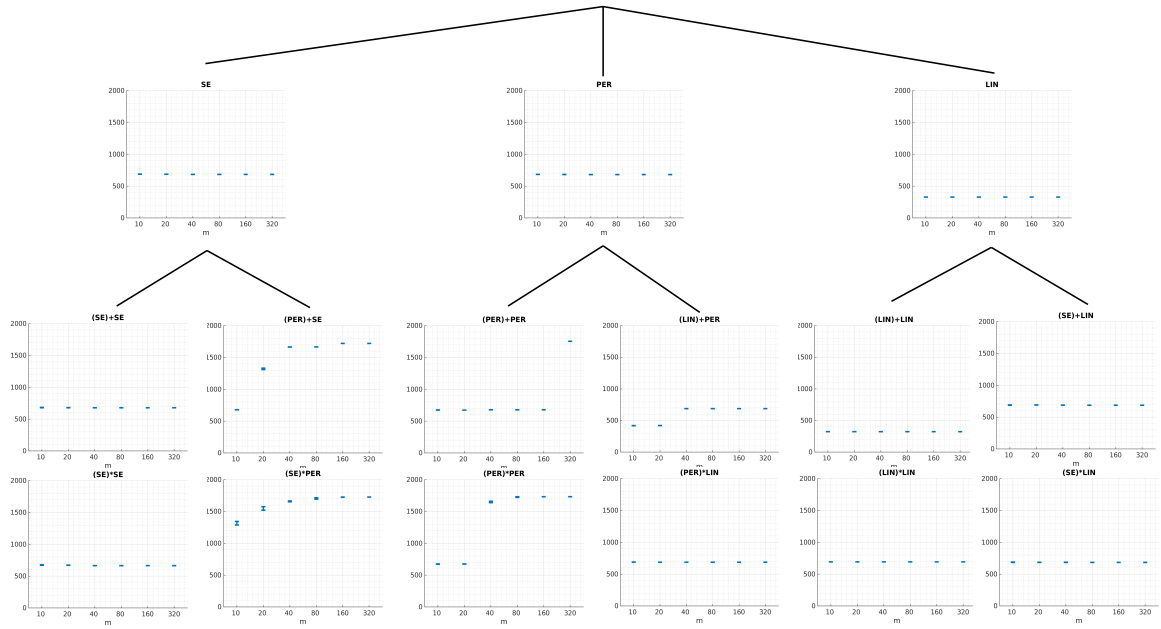


Figure 4.13: Same as Figure 4.12 but for Mauna data.

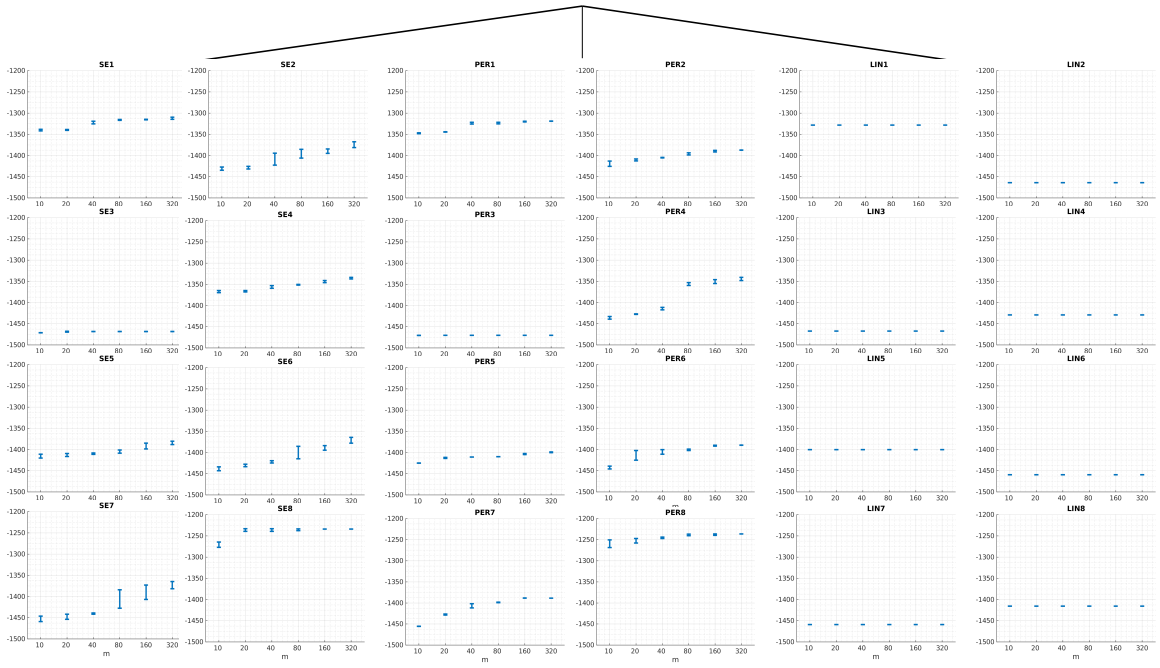


Figure 4.14: Same as Figure 4.12 but for concrete data and up to depth 1.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Scaling up the Automatic Statistician: Scalable Structure Discovery using Gaussian Processes
Publication Status	x Published □ Accepted for Publication □Submitted for Publication □ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Kim, H. and Teh, Y.W., 2018, April. Scaling up the Automatic Statistician: Scalable Structure Discovery using Gaussian Processes. In Artificial Intelligence and Statistics (AISTATS).

Student Confirmation

Student Name:	Hyun Jik Kim		

Contribution to the Paper	- Solo lead of the project, with supervisor as single advisor.
	- Formulated the problem that the paper tackles
	- Came up with all central ideas for tackling the problem, including:
	  - Derivation and implementation of the upper bound
	  - Design of the kernel search algorithm
	- Implemented the algorithm in code and carried out all experiments.
	- All of the paper writing (with feedback from advisors).		

Signature		Date	

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Yee Whye Teh, Professor of Statistical Machine Learning		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

Chapter 5

Disentangling by Factorising

The following chapter is a self-contained paper:

> **Kim, H.** and Mnih, A., 2018, July. Disentangling by Factorising. In Proceedings of the 35th International Conference on Machine Learning (ICML).

There is a statement of authorship at the end of this chapter.

Abstract

We define and address the problem of unsupervised learning of disentangled representations on data generated from independent factors of variation. We propose FactorVAE, a method that disentangles by encouraging the distribution of representations to be factorial and hence independent across the dimensions. We show that it improves upon β-VAE by providing a better trade-off between disentanglement and reconstruction quality. Moreover, we highlight the problems of a commonly used disentanglement metric and introduce a new metric that does not suffer from them.

5.1 Introduction

Learning interpretable representations of data that expose semantic meaning has important consequences for artificial intelligence. Such representations are useful not only for standard downstream tasks such as supervised learning and reinforcement learning, but also for tasks such as transfer learning and zero-shot learning where humans excel but machines struggle [Lake et al., 2016]. There have been multiple

efforts in the deep learning community towards learning factors of variation in the data, commonly referred to as learning a *disentangled representation*. While there is no canonical definition for this term, we adopt the one due to Bengio et al. [2013]: a representation where a change in one dimension corresponds to a change in one factor of variation, while being relatively invariant to changes in other factors. In particular, we assume that the data has been generated from a fixed number of independent factors of variation.[3] We focus on image data, where the effect of factors of variation is easy to visualise.

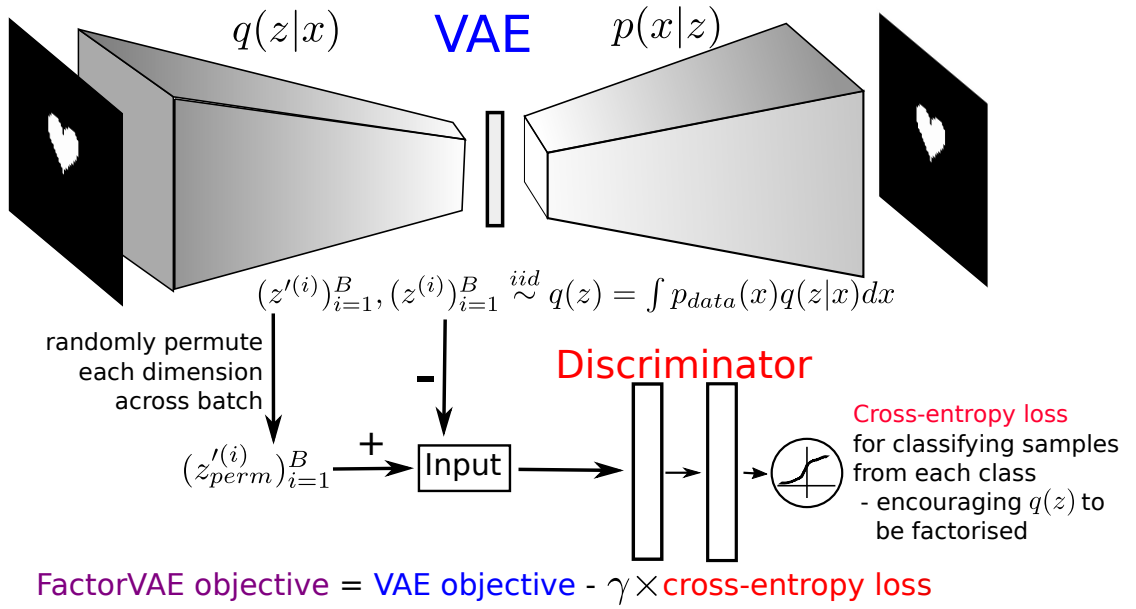


Figure 5.1: Architecture of FactorVAE, a Variational Autoencoder (VAE) that encourages the code distribution to be factorial. The top row is a VAE with convolutional encoder and decoder, and the bottom row is a MLP classifier, the discriminator, that distinguishes whether the input was drawn from the marginal code distribution or the product of its marginals.

Using generative models has shown great promise in learning disentangled representations in images. Notably, semi-supervised approaches that require implicit or explicit knowledge about the true underlying factors of the data have excelled at disentangling [Kulkarni et al., 2015, Kingma et al., 2014, Reed et al., 2014, Siddharth et al., 2017, Hinton et al., 2011, Mathieu et al., 2016, Goroshin et al., 2015, Hsu et al., 2017, Denton and Birodkar, 2017]. However, ideally we would like to learn these in an unsupervised manner, due to the following reasons: 1. Humans are able to learn factors of variation

[3]We discuss the limitations of this assumption in Section 5.4.

unsupervised [Perry et al., 2010]. 2. Labels are costly as obtaining them requires a human in the loop. 3. Labels assigned by humans might be inconsistent or leave out the factors that are difficult for humans to identify.

β-VAE [Higgins et al., 2016] is a popular method for unsupervised disentangling based on the Variational Autoencoder (VAE) framework [Kingma and Welling, 2013, Rezende et al., 2014] for generative modelling. It uses a modified version of the VAE objective with a larger weight ($\beta > 1$) on the KL divergence between the variational posterior and the prior, and has empirically demonstrated the possibility of unsupervised disentangling on image data. One drawback of β-VAE is that reconstruction quality (compared to VAE) must be sacrificed in order to obtain better disentangling. The goal of our work is to obtain a better trade-off between disentanglement and reconstruction, allowing to achieve better disentanglement without degrading reconstruction quality. In this work, we analyse the source of this trade-off and propose FactorVAE, which augments the VAE objective with a penalty that encourages the marginal distribution of representations to be factorial without substantially affecting the quality of reconstructions. This penalty is expressed as a KL divergence between this marginal distribution and the product of its marginals, and is optimised using a discriminator network following the divergence minimisation view of GANs [Nowozin et al., 2016]. Our experimental results show that this approach achieves better disentanglement than β-VAE for the same reconstruction quality. In addition, we point out the weaknesses in the disentangling metric of Higgins et al. [2016], and propose a new metric that addresses these shortcomings.

A popular alternative to β-VAE is InfoGAN [Chen et al., 2016], which is based on the Generative Adversarial Net (GAN) framework [Goodfellow et al., 2014a] for generative modelling. InfoGAN learns disentangled representations by rewarding the mutual information between the observations and a subset of latents. However at least in part due to its training stability issues [Higgins et al., 2016], there has been little empirical comparison between VAE-based methods and InfoGAN. Taking advantage of the recent developments in the GAN literature that help stabilise training, we include InfoWGAN-GP, a version of InfoGAN that uses Wasserstein distance [Arjovsky et al., 2017] and gradient penalty [Gulrajani et al., 2017], in our experimental evaluation.

In summary, we make the following contributions: 1) We introduce FactorVAE, a method for disentangling that gives higher disentanglement scores than β-VAE for the same reconstruction quality. 2) We identify the weaknesses of the disentanglement metric of Higgins et al. [2016] and propose a more robust alternative. 3) We give

quantitative comparisons of FactorVAE and β-VAE against InfoGAN's WGAN-GP counterpart for disentanglement.

5.2 Trade-off between Disentanglement and Reconstruction in β-VAE

We motivate our approach by analysing where the disentanglement and reconstruction trade-off arises in the β-VAE objective. First, we introduce notation and architecture of our VAE framework. We assume that observations $x^{(i)} \in \mathcal{X}, i = 1, \ldots, N$ are generated by combining K underlying factors $f = (f_1, \ldots, f_K)$. These observations are modelled using a real-valued latent/code vector $z \in \mathbb{R}^d$, interpreted as the representation of the data. The generative model is defined by the standard Gaussian prior $p(z) = \mathcal{N}(0, I)$, intentionally chosen to be a factorised distribution, and the decoder $p_\theta(x|z)$ parameterised by a neural net. The variational posterior for an observation is $q_\theta(z|x) = \prod_{j=1}^{d} \mathcal{N}(z_j|\mu_j(x), \sigma_j^2(x))$, with the mean and variance produced by the encoder, also parameterised by a neural net.[1] The variational posterior can be seen as the distribution of the representation corresponding to the data point x. The distribution of representations for the entire data set is then given by

$$q(z) = \mathbb{E}_{p_{data}(x)}[q(z|x)] = \frac{1}{N} \sum_{i=1}^{N} q(z|x^{(i)}), \tag{5.1}$$

which is known as the marginal posterior or aggregate posterior [Tang et al., 2012], where p_{data} is the empirical data distribution. A disentangled representation would have each z_j correspond to precisely one underlying factor f_k. Since we assume that these factors vary independently, we wish for a factorial distribution $q(z) = \prod_{j=1}^{d} q(z_j)$.

The β-VAE objective

$$\frac{1}{N} \sum_{i=1}^{N} \left[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - \beta KL(q(z|x^{(i)})||p(z)) \right]$$

is a variational lower bound on $\mathbb{E}_{p_{data}(x)}[\log p(x^{(i)})]$ for $\beta \geq 1$, reducing to the VAE objective for $\beta = 1$. Its first term can be interpreted as the negative *reconstruction error*, and the second term as the complexity penalty that acts as a regulariser. We

[1] In the rest of the paper we will omit the dependence of p and q on their parameters θ for notational convenience.

may further break down this KL term as [Hoffman and Johnson, 2016, Makhzani and Frey, 2017]

$$\mathbb{E}_{p_{data}(x)}[KL(q(z|x)||p(z))] = I(x;z) + KL(q(z)||p(z)),$$

where $I(x;z)$ is the mutual information between x and z under the joint distribution $p_{data}(x)q(z|x)$. See Appendix 5.8.3 for the derivation. Penalising the $KL(q(z)||p(z))$ term pushes $q(z)$ towards the factorial prior $p(z)$, encouraging independence in the dimensions of z, a necessary condition for disentangling when the factors of variation are independent. Penalising $I(x;z)$, on the other hand, reduces the amount of information about x stored in z, which can lead to poor reconstructions for high values of β [Makhzani and Frey, 2017]. Thus making β larger than 1, penalising both terms more, leads to better disentanglement but reduces reconstruction quality. When this reduction is severe, there is insufficient information about the observation in the latents, making it impossible to recover the true factors. Therefore there exists a value of $\beta > 1$ that gives highest disentanglement, but results in a higher reconstruction error than a VAE.

5.3 Total Correlation Penalty and FactorVAE

Penalising $I(x;z)$ more than a VAE does might be neither necessary nor desirable for disentangling. For example, InfoGAN disentangles by encouraging $I(x;c)$ to be high where c is a subset of the latent variables z [2]. Hence we motivate FactorVAE by augmenting the VAE objective with a term that directly encourages independence in the code distribution, arriving at the following objective:

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - KL(q(z|x^{(i)})||p(z))\Big] - \gamma KL(q(z)||\bar{q}(z)) \tag{5.2}$$

where $\bar{q}(z) \coloneqq \prod_{j=1}^{d} q(z_j)$. Note that this is also a lower bound on the marginal log likelihood $\mathbb{E}_{p_{data}(x)}[\log p(x)]$. $KL(q(z)||\bar{q}(z))$ is known as *Total Correlation* (TC) [Watanabe, 1960], a popular measure of dependence for multiple random variables. In our case this term is intractable since both $q(z)$ and $\bar{q}(z)$ involve mixtures with a large number of components, and the direct Monte Carlo estimate requires a pass through the entire data set for each $q(z)$ evaluation.[3]. Hence we take an alternative

[2]Note however that $I(x;z)$ in β-VAE is defined under the joint distribution of data and their encoding distribution $p_{data}(x)q(z|x)$, whereas $I(x;c)$ in InfoGAN is defined on the joint distribution of the prior on c and the decoding distribution $p(c)p(x|c)$.

[3]We have also tried using a batch estimate of $q(z)$, but this did not work. See Appendix 5.8.4 for details.

approach for optimizing this term. We start by observing we can sample from $q(z)$ efficiently by first choosing a datapoint $x^{(i)}$ uniformly at random and then sampling from $q(z|x^{(i)})$. We can also sample from $\bar{q}(z)$ by generating d samples from $q(z)$ and then ignoring all but one dimension for each sample. A more efficient alternative involves sampling a batch from $q(z)$ and then randomly permuting across the batch for each latent dimension (see Algorithm 3). This is a standard trick used in the independence testing literature [Arcones and Gine, 1992] and as long as the batch is large enough, the distribution of these samples will closely approximate $\bar{q}(z)$.

Having access to samples from both distributions allows us to minimise their KL divergence using the *density-ratio trick* [Nguyen et al., 2010, Sugiyama et al., 2012] which involves training a classifier/discriminator to approximate the density ratio that arises in the KL term. Suppose we have a discriminator D (in our case an MLP) that outputs an estimate of the probability $D(z)$ that its input is a sample from $q(z)$ rather than from $\bar{q}(z)$. Then we have

$$TC(z) = KL(q(z)||\bar{q}(z)) = \mathbb{E}_{q(z)}\left[\log \frac{q(z)}{\bar{q}(z)}\right] \approx \mathbb{E}_{q(z)}\left[\log \frac{D(z)}{1 - D(z)}\right]. \tag{5.3}$$

We train the discriminator and the VAE jointly. In particular, the VAE parameters are updated using the objective in Equation (5.2), with the TC term replaced using the discriminator-based approximation from Equation (5.3). The discriminator is trained to classify between samples from $q(z)$ and $\bar{q}(z)$, thus learning to approximate the density ratio needed for estimating TC. See Algorithm 4 for pseudocode of FactorVAE.

Algorithm 3 `permute_dims`

Input: $\{z^{(i)} \in \mathbb{R}^d : i = 1, \ldots, B\}$
for $j = 1$ **to** d **do**
  $\pi \leftarrow$ random permutation on $\{1, \ldots, B\}$
  $(z_j^{(i)})_{i=1}^B \leftarrow (z_j^{(\pi(i))})_{i=1}^B$
end for
Output: $\{z^{(i)} : i = 1, \ldots, B\}$

It is important to note that low TC is necessary but not sufficient for meaningful disentangling. For example, when $q(z|x) = p(z)$, TC=0 but z carries no information about the data. Thus having low TC is only meaningful when we can preserve information in the latents, which is why controlling for reconstruction error is important. Also note that even when inference is perfect, i.e. $q(z|x) = p(z|x)$, this does not necessarily imply $q(z) = p(z)$, since $q(z) = \int q(z|x)p_{data}(x)dx$ whereas $p(z) = \int p(z|x)p(x)dx$ where $p(x)$ is the marginal distribution of samples generated from the model. For the

Algorithm 4 FactorVAE

Input: observations $(x^{(i)})_{i=1}^{N}$, batch size m, latent dimension d, γ, VAE/Discriminator optimisers: g, g_D
Initialize VAE and discriminator parameters θ, ψ.
repeat
  Randomly select batch $(x^{(i)})_{i\in\mathcal{B}}$ of size m
  Sample $z_\theta^{(i)} \sim q_\theta(z|x^{(i)})\ \forall i \in \mathcal{B}$
  $\theta \leftarrow g(\nabla_\theta \frac{1}{m} \sum_{i\in\mathcal{B}} [\log \frac{p_\theta(x^{(i)}, z_\theta^{(i)})}{q_\theta(z_\theta^{(i)}|x^{(i)})} - \gamma \log \frac{D_\psi(z_\theta^{(i)})}{1-D_\psi(z_\theta^{(i)})}])$
  Randomly select batch $(x^{(i)})_{i\in\mathcal{B}'}$ of size m
  Sample $z_\theta'^{(i)} \sim q_\theta(z|x^{(i)})$ for $i \in \mathcal{B}'$
  $(z_{perm}'^{(i)})_{i\in\mathcal{B}'} \leftarrow \texttt{permute\_dims}((z_\theta'^{(i)})_{i\in\mathcal{B}'})$
  $\psi \leftarrow g_D(\nabla_\psi \frac{1}{2m} [\sum_{i\in\mathcal{B}} \log(D_\psi(z_\theta^{(i)}))$
    $+ \sum_{i\in\mathcal{B}'} \log(1 - D_\psi(z_{perm}'^{(i)}))])$
until convergence of objective.

aggregate posterior to match the prior, the model distribution must also have fit the true data distribution, a difficult task in practice. Hence the TC penalty is still valid for VAE models with richer inference networks or flow-based generative models Dinh et al. [2016], where inference is exact.

In the GAN literature, divergence minimisation is usually done between two distributions over the data space, which is often very high dimensional (e.g. images). As a result, the two distributions often have disjoint support, making training unstable, especially when the discriminator is strong. Hence it is necessary to use tricks to weaken the discriminator such as instance noise [Sønderby et al., 2016] or to replace the discriminator with a critic, as in Wasserstein GANs [Arjovsky et al., 2017]. In this work, we minimise divergence between two distributions over the latent space (as in e.g. Mescheder et al. [2017]), which is typically much lower dimensional and the two distributions have overlapping support. We observe that training is stable for sufficiently large batch sizes (e.g. 64 worked well for $d = 10$), allowing us to use a strong discriminator.

5.4 A New Metric for Disentanglement

The definition of disentanglement we use in this paper, where a change in one dimension of the representation corresponds to a change in exactly one factor of variation, is

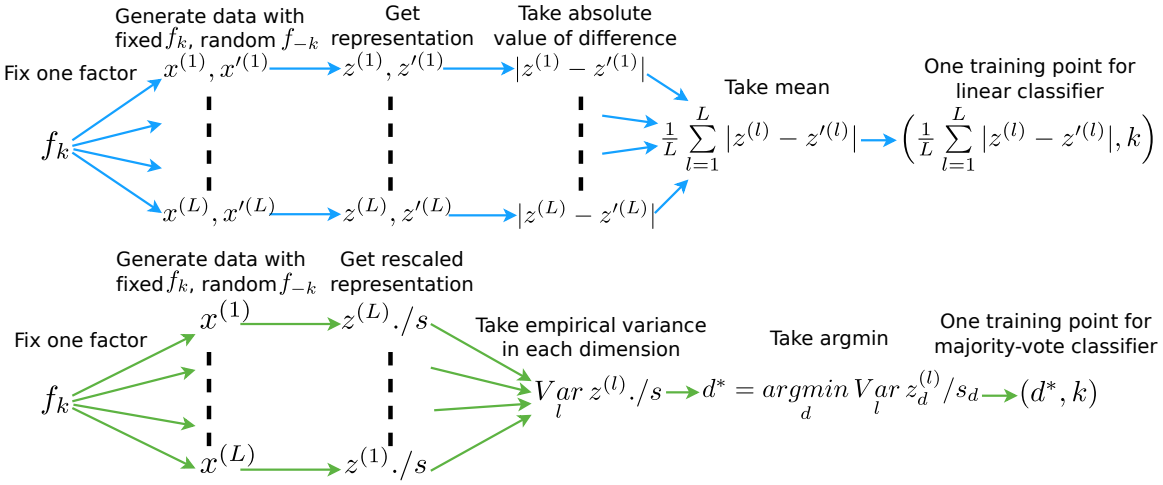


Figure 5.2: Top: Metric in Higgins et al. [2016]. Bottom: Our new metric, where $s \in \mathbb{R}^d$ is the scale (empirical standard deviation) of latent representations of the full data (or large enough random subset).

clearly a simplistic one. It does not allow correlations among the factors or hierarchies over them. Thus this definition seems more suited to synthetic data with independent factors of variation than to most realistic data sets. However, as we will show below, robust disentanglement is not a fully solved problem even in this simple setting. One obstacle on the way to this first milestone is the absence of a sound quantitative metric for measuring disentanglement.

A popular method of measuring disentanglement is by inspecting *latent traversals*: visualising the change in reconstructions while traversing one dimension of the latent space at a time. Although latent traversals can be a useful indicator of when a model has failed to disentangle, the qualitative nature of this approach makes it unsuitable for comparing algorithms reliably. Doing this would require inspecting a multitude of latent traversals over multiple reference images, random seeds, and points during training. Having a human in the loop to assess the traversals is also too time-consuming and subjective. Unfortunately, for data sets that do not have the ground truth factors of variation available, currently this is the only viable option for assessing disentanglement.

Higgins et al. [2016] proposed a supervised metric that attempts to quantify disentanglement when the ground truth factors of a data set are given. The metric is the error rate of a linear classifier that is trained as follows. Choose a factor k; generate data with this factor fixed but all other factors varying randomly; obtain their representations (defined to be the mean of $q(z|x)$); take the absolute value of

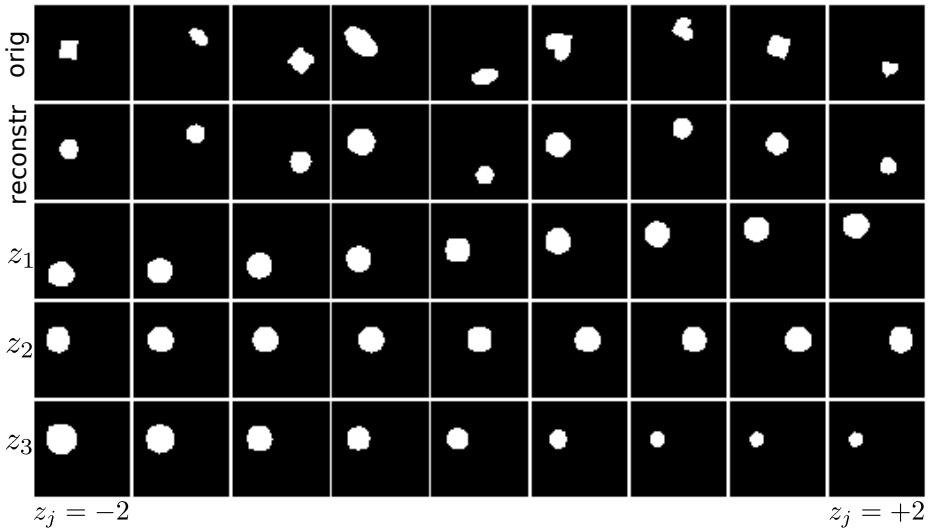


Figure 5.3: A β-VAE model trained on the 2D Shapes data that scores 100% on metric in Higgins et al. [2016] (ignoring the shape factor). First row: originals. Second row: reconstructions. Remaining rows: reconstructions of latent traversals. The model only uses three latent units to capture x-position, y-position, scale and ignores orientation, yet achieves a perfect score on the metric.

the pairwise differences of these representations. Then the mean of these statistics across the pairs gives one training input for the classifier, and the fixed factor index k is the corresponding training output (see top of Figure 5.2). So if the representations were perfectly disentangled, we would see zeros in the dimension of the training input that corresponds to the fixed factor of variation, and the classifier would learn to map the index of the zero value to the index of the factor.

However this metric has several weaknesses. Firstly, it could be sensitive to hyperparameters of the linear classifier optimisation, such as the choice of the optimiser and its hyperparameters, weight initialisation, and the number of training iterations. Secondly, having a linear classifier is not so intuitive – we could get representations where each factor corresponds to a linear combination of dimensions instead of a single dimension. Finally and most importantly, the metric has a failure mode: it gives 100% accuracy even when only $K - 1$ factors out of K have been disentangled; to predict the remaining factor, the classifier simply learns to detect when all the values corresponding to the $K - 1$ factors are non-zero. An example of such a case is shown in Figure 5.3.

To address these weaknesses, we propose a new disentanglement metric as follows. Choose a factor k; generate data with this factor fixed but all other factors varying randomly; obtain their representations; normalise each dimension by its empirical

standard deviation over the full data (or a large enough random subset); take the empirical variance in each dimension[4] of these normalised representations. Then the index of the dimension with the lowest variance and the target index k provide one training input/output example for the classifier (see bottom of Figure 5.2). Thus if the representation is perfectly disentangled, the empirical variance in the dimension corresponding to the fixed factor will be 0. We normalise the representations so that the arg min is invariant to rescaling of the representations in each dimension. Since both inputs and outputs lie in a discrete space, the optimal classifier is the majority-vote classifier (see Appendix 5.8.2 for details), and the metric is the error rate of the classifier. The resulting classifier is a deterministic function of the training data, hence there are no optimisation hyperparameters to tune. We also believe that this metric is conceptually simpler and more natural than the previous one. Most importantly, it circumvents the failure mode of the earlier metric, since the classifier needs to see the lowest variance in a latent dimension for a given factor to classify it correctly.

We think developing a reliable unsupervised disentangling metric that does not use the ground truth factors is an important direction for future research, since unsupervised disentangling is precisely useful for the scenario where we do not have access to the ground truth factors. With this in mind, we believe that having a reliable supervised metric is still valuable as it can serve as a gold standard for evaluating unsupervised metrics.

5.5 Related Work

There are several recent works that use a discriminator to optimise a divergence to encourage independence in the latent codes. Adversarial Autoencoder [AAE, Makhzani et al., 2015] removes the $I(x; z)$ term in the VAE objective and maximizes the negative reconstruction error minus $KL(q(z)||p(z))$ using the density-ratio trick. This means that the AAE objective is not a lower bound on the log marginal likelihood. Moreover, the emphasis of that work is on semi-supervised classification and unsupervised clustering, rather than disentangling. In PixelGAN Autoencoders [Makhzani and Frey, 2017], the same objective is used to study the decomposition of information between the latent code and the decoder. The authors state that adding noise to the inputs

[4]We can use Gini's definition of variance for discrete latents [Gini, 1971]. See Appendix 5.8.2 for details.

of the encoder is crucial, which suggests that limiting the information that the code contains about the input is essential and that the $I(x; z)$ term should not be dropped from the VAE objective. Brakel and Bengio [2017] also use a discriminator to penalise the Jensen-Shannon Divergence between the distribution of codes and the product of its marginals. However, they use the GAN loss with deterministic encoders and decoders and only explore their technique in the context of Independent Component Analysis source separation.

Early works on unsupervised disentangling include [Schmidhuber, 1992] which attempts to disentangle codes in an autoencoder by penalising predictability of one latent dimension given the others and [Desjardins et al., 2012] where a variant of a Boltzmann Machine is used to disentangle two factors of variation in the data. More recently, Achille and Soatto [2018] have used a loss function that penalises TC in the context of supervised learning. They show that their approach can be extended to the VAE setting, but do not perform any experiments on disentangling to support the theory. In a concurrent work, Kumar et al. [2018a] used moment matching in VAEs to penalise the covariance between the latent dimensions, but did not constrain the mean or higher moments. We provide the objectives used in these related methods and analyse them in Appendix 5.9.1.

There have been various works that use the notion of predictability to quantify disentanglement, mostly predicting the value of ground truth factors $f = (f_1, \ldots, f_K)$ from the latent code z. This dates back to Yang and Amari [1997] who learn a linear map from representations to factors in the context of linear ICA, and quantify how close this map is to a permutation matrix. More recently Eastwood and Williams [2018] have extended this idea to disentanglement by training a Lasso regressor to map z to f and using its trained weights to quantify disentanglement. Like other regression-based approaches, this one introduces hyperparameters such as the optimiser and the Lasso penalty coefficient. The metric of Higgins et al. [2016] as well as the one we proposed, predict the factor k from the z of images with a fixed f_k but f_{-k} varying randomly. Schmidhuber [1992] quantifies predictability between the different dimensions of z, using a predictor that is trained to predict z_j from z_{-j}.

Invariance and *equivariance* are frequently considered to be desirable properties of representations in the literature [Goodfellow et al., 2009, Kivinen and Williams, 2011, Lenc and Vedaldi, 2015]. A representation is said to be *invariant* for a particular task if it does not change when nuisance factors of the data, that are irrelevant to the task, are changed. An *equivariant* representation changes in a stable and predictable

manner when altering a factor of variation. A disentangled representation, in the sense used in the paper, is equivariant, since changing one factor of variation will change one dimension of a disentangled representation in a predictable manner. Given a task, it will be easy to obtain an invariant representation from the disentangled representation by ignoring the dimensions encoding the nuisance factors for the task [Cohen and Welling, 2014].

Building on a preliminary version of this work, Chen et al. [2018] recently proposed a minibatch-based alternative to our density-ratio-trick-based method for estimating the Total Correlation and introduced an information-theoretic disentangling metric.

5.6 Experiments

We compare FactorVAE to β-VAE on the following data sets with i) known generative factors: 1) **2D Shapes** [Matthey et al., 2017]: 737,280 binary 64×64 images of 2D shapes with ground truth factors[number of values]: shape[3], scale[6], orientation[40], x-position[32], y-position[32]. 2) **3D Shapes** [Burgess and Kim, 2018] data: 480,000 RGB $64 \times 64 \times 3$ images of 3D shapes with ground truth factors: shape[4], scale[8], orientation[15], floor colour[10], wall colour[10], object colour[10] ii) unknown generative factors: 3) **3D Faces** [Paysan et al., 2009]: 239,840 grey-scale 64×64 images of 3D Faces. 4) **3D Chairs** [Aubry et al., 2014]: 86,366 RGB $64 \times 64 \times 3$ images of chair CAD models. 5) **CelebA** (cropped version) [Liu et al., 2015]: 202,599 RGB $64 \times 64 \times 3$ images of celebrity faces. The experimental details such as encoder/decoder architectures and hyperparameter settings are in Appendix 5.8.1. The details of the disentanglement metrics, along with a sensitivity analysis with respect to their hyperparameters, are given in Appendix 5.8.2.

From Figure 5.4, we see that FactorVAE gives much better disentanglement scores than VAEs ($\beta = 1$), while barely sacrificing reconstruction error, highlighting the disentangling effect of adding the Total Correlation penalty to the VAE objective. The best disentanglement scores for FactorVAE are noticeably better than those for β-VAE given the same reconstruction error. This can be seen more clearly in Figure 5.5 where the best mean disentanglement of FactorVAE ($\gamma = 40$) is around 0.82, significantly higher than the one for β-VAE ($\beta = 4$), which is around 0.73, both with reconstruction error around 45. From Figure 5.6, we can see that both models are capable of finding x-position, y-position, and scale, but struggle to disentangle orientation and shape,

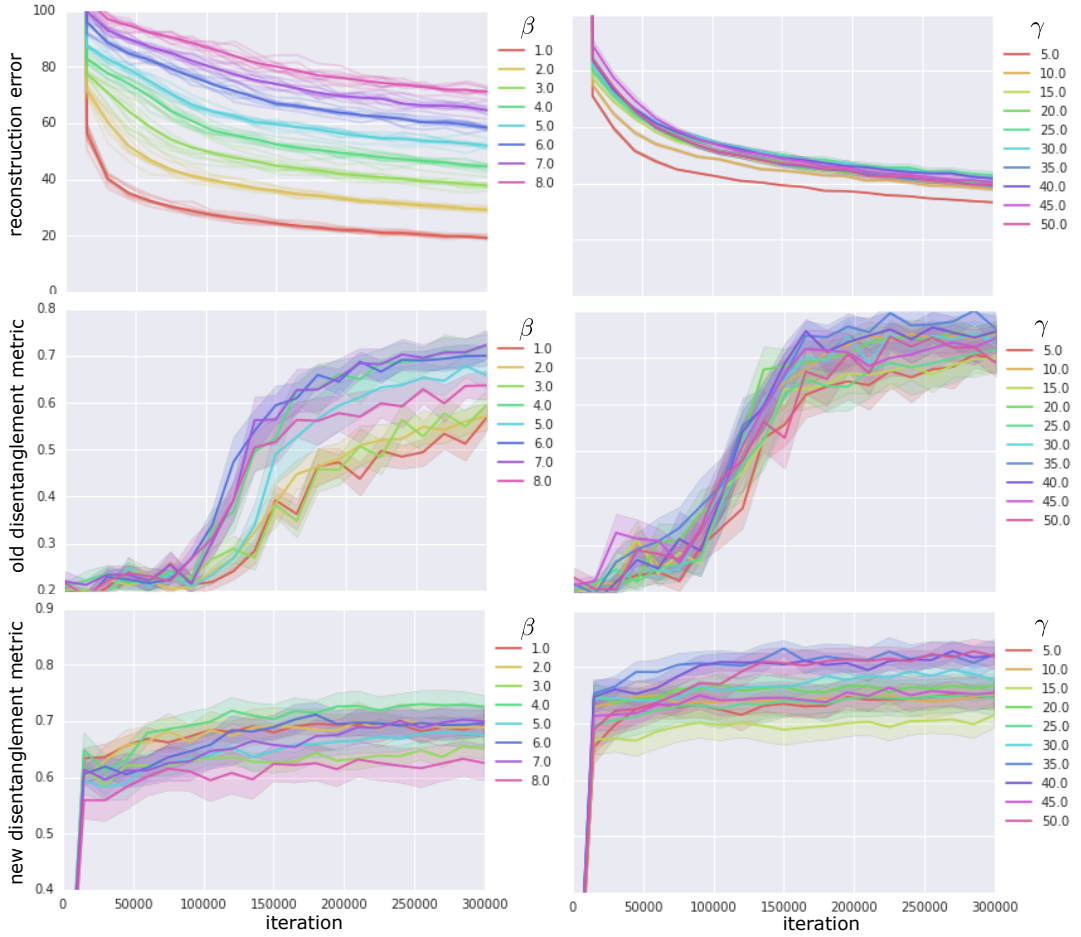


Figure 5.4: Reconstruction error (top), metric in Higgins et al. [2016] (middle), our metric (bottom). β-VAE (left), FactorVAE (right). The colours correspond to different values of β and γ respectively, and confidence intervals are over 10 random seeds.

β-VAE especially. For this data set, neither method can robustly capture shape, the discrete factor of variation[5].

As a sanity check, we also evaluated the correlation between our metric and the metric in Higgins et al. [2016]: **Pearson** (linear correlation coefficient): 0.404, **Kendall** (proportion of pairs that have the same ordering): 0.310, **Spearman** (linear correlation of the rankings): 0.444, all with p-value 0.000. Hence the two metrics show a fairly high positive correlation as expected.

We have also examined how the discriminator's estimate of the Total Correlation (TC) behaves and the effect of γ on the true TC. From Figure 5.7, observe that the

[5]This is partly due to the fact that learning discrete factors would require using discrete latent variables instead of Gaussians, but jointly modelling discrete and continuous factors of variation is a non-trivial problem that needs further research.

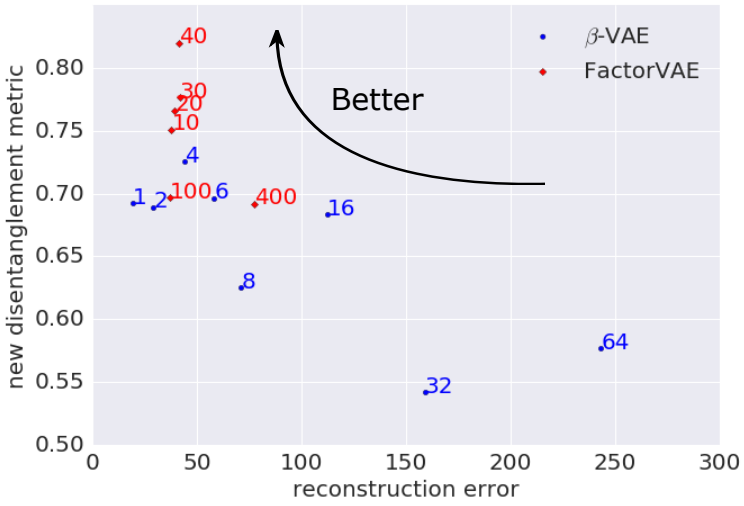


Figure 5.5: Reconstruction error plotted against our disentanglement metric, both averaged over 10 random seeds at the end of training. The numbers at each point are values of β and γ. Note that we want low reconstruction error and a high disentanglement metric.

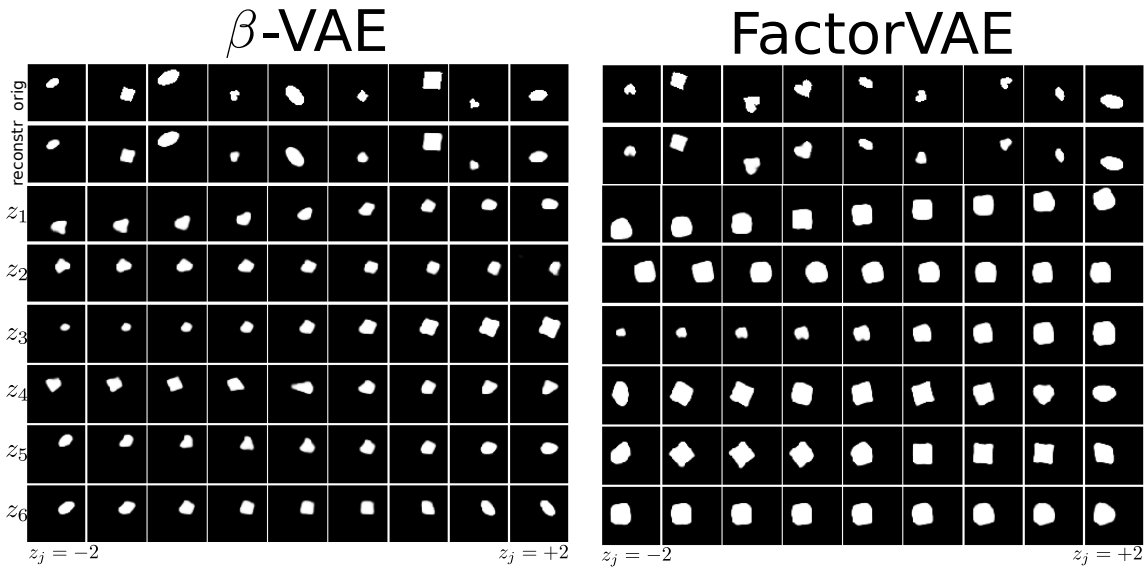


Figure 5.6: First row: originals. Second row: reconstructions. Remaining rows: reconstructions of latent traversals across each latent dimension sorted by $KL(q(z_j|x)||p(z_j))$, for the best scoring models on our disentanglement metric. Left: β-VAE, score: 0.814, $\beta = 4$. Right: FactorVAE, score: 0.889, $\gamma = 35$.

discriminator is consistently underestimating the true TC, also confirmed in Rosca et al. [2018]. However the true TC decreases throughout training, and a higher γ leads to lower TC, so the gradients obtained using the discriminator are sufficient for encouraging independence in the code distribution.

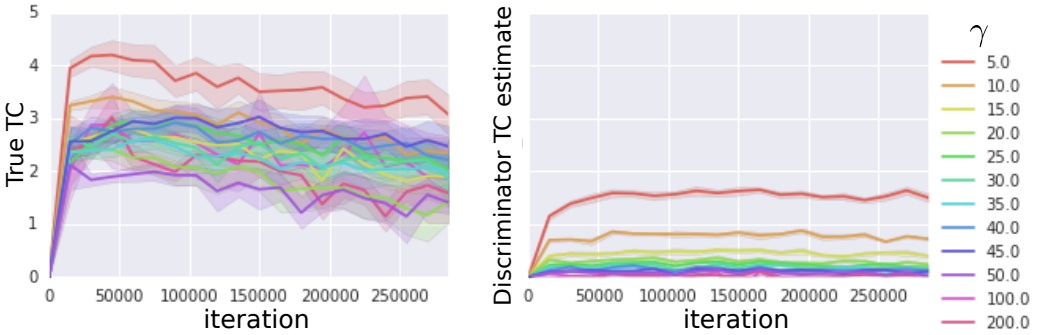


Figure 5.7: Total Correlation values for FactorVAE on 2D Shapes. Left: True TC value. Right: Discriminator's estimate of TC.

We then evaluated InfoWGAN-GP, the counterpart of InfoGAN that uses Wasserstein distance and gradient penalty. See Appendix 5.9.2 for an overview. One advantage of InfoGAN is that the Monte Carlo estimate of its objective is differentiable with respect to its parameters even for discrete codes c, which makes gradient-based optimisation straightforward. In contrast, VAE-based methods that rely on the reparameterisation trick for gradient-based optimisation require z to be a reparameterisable continuous random variable and alternative approaches require various variance reduction techniques for gradient estimation [Mnih and Rezende, 2016a, Maddison et al., 2016]. Thus we might expect Info(W)GAN(-GP) to show better disentangling in cases where some factors are discrete. Hence we use 4 continuous latents (one for each continuous factor) and one categorical latent of 3 categories (one for each shape). We tuned for λ, the weight of the mutual information term in Info(W)GAN(-GP), $\in \{0.0, 0.1, 0.2, \ldots, 1.0\}$, number of noise variables $\in \{5, 10, 20, 40, 80, 160\}$ and the learning rates of the generator $\in \{10^{-3}, 10^{-4}\}$, discriminator $\in \{10^{-4}, 10^{-5}\}$.

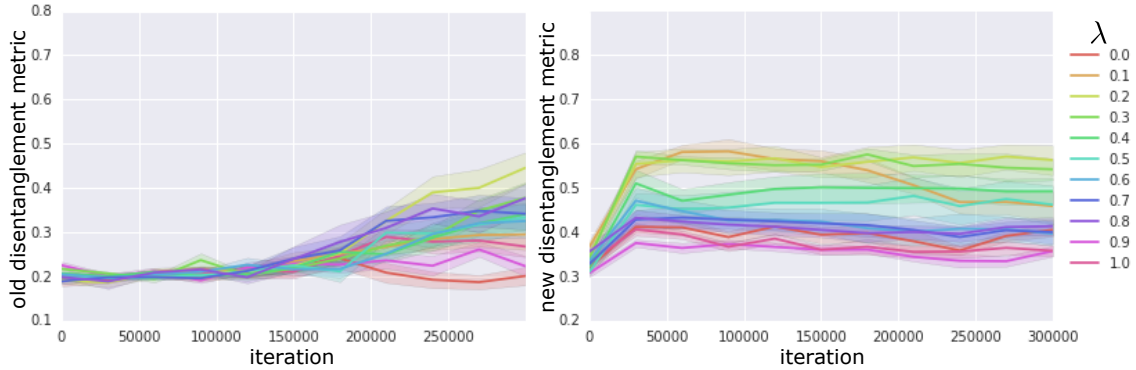


Figure 5.8: Disentanglement scores for InfoWGAN-GP on 2D Shapes for 10 random seeds per hyperparameter setting. Left: Metric in Higgins et al. [2016]. Right: Our metric.

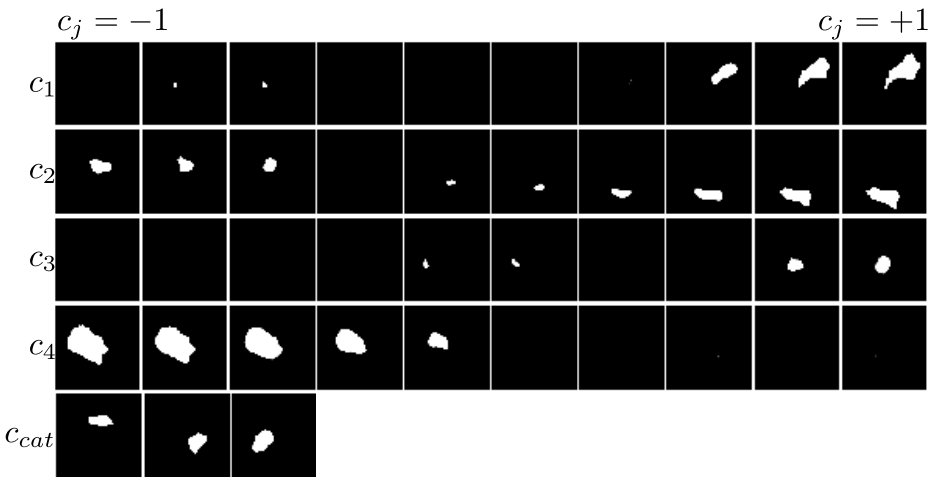


Figure 5.9: Latent traversals for InfoWGAN-GP on 2D Shapes across four continuous codes (first four rows) and categorical code (last row) for run with best disentanglement score ($\lambda = 0.2$).

However from Figure 5.8 we can see that the disentanglement scores are disappointingly low. From the latent traversals in Figure 5.9, we can see that the model learns only the scale factor, and tries to put positional information in the discrete latent code, which is one reason for the low disentanglement score. Using 5 continuous codes and no categorical codes did not improve the disentanglement scores however. InfoGAN with early stopping (before training instability occurs – see Appendix 5.9.3) also gave similar results. The fact that some latent traversals give blank reconstructions indicates that the model does not generalise well to all parts of the domain of $p(z)$.

One reason InfoWGAN-GP's poor performance on this data set could be that InfoGAN is sensitive to the generator and discriminator architecture, which is one thing we did not tune extensively. We use a similar architecture to the VAE-based approaches for 2D shapes for a fair comparison, but have also tried a bigger architecture which gave similar results (see Appendix 5.9.3). If architecture search is indeed important, this would be a weakness of InfoGAN relative to FactorVAE and β-VAE, which are both much more robust to architecture choice. In Appendix 5.9.3, we check that we can replicate the results of Chen et al. [2016] on MNIST using InfoWGAN-GP, verify that it makes training stable compared to InfoGAN, and give implementation details with further empirical studies of InfoGAN and InfoWGAN-GP.

We now show results on the 3D Shapes data, which is a more complex data set of 3D scenes with additional features such as shadows and background (sky). We train both β-VAE and FactorVAE for 1M iterations. Figure 5.10 again shows that FactorVAE

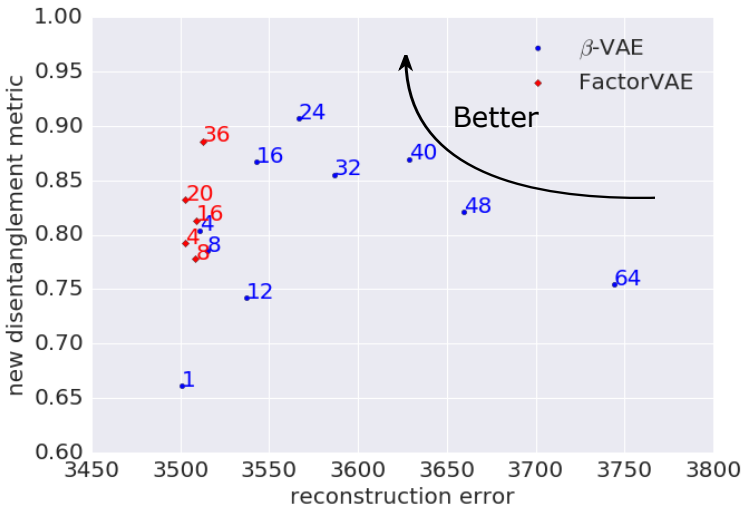


Figure 5.10: Same as Figure 5.5 for 3D Shapes data, 5 random seeds.

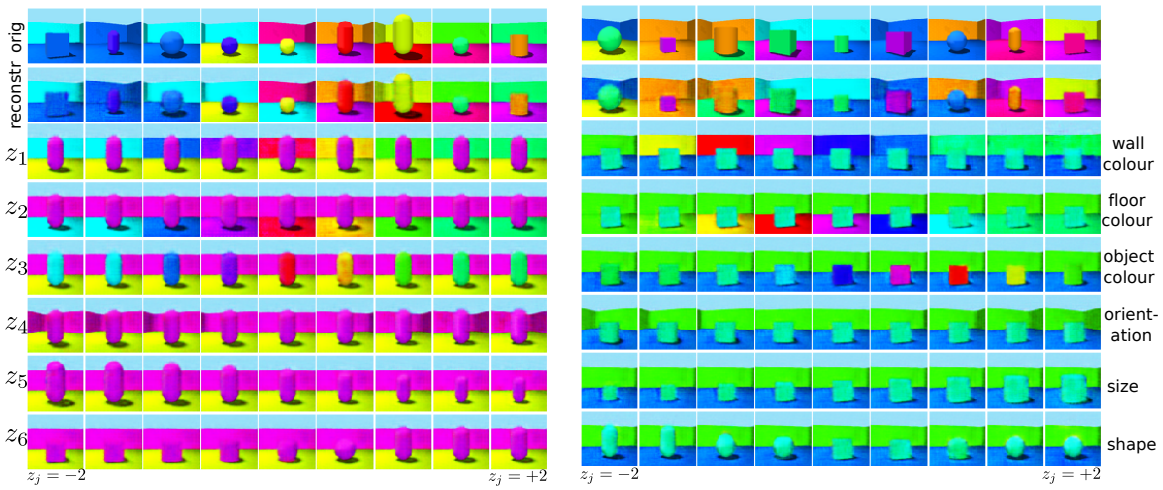


Figure 5.11: Same as Figure 5.6 but for 3D Shapes data. Left: β-VAE, score: 1.00, $\beta = 32$. Right: FactorVAE, score: 1.00, $\gamma = 7$.

achieves much better disentanglement with barely any increase in reconstruction error compared to VAE. Moreover, while the top mean disentanglement scores for FactorVAE and β-VAE are similar, the reconstruction error is lower for FactorVAE: 3515 ($\gamma = 36$) as compared to 3570 ($\beta = 24$). The latent traversals in Figure 5.11 show that both models are able to capture the factors of variation in the best-case scenario. Looking at latent traversals across many random seeds, however, makes it evident that both models struggled to disentangle the factors for shape and scale.

To show that FactorVAE also gives a valid generative model for both 2D Shapes and 3D Shapes, we present the log marginal likelihood evaluated on the entire data set

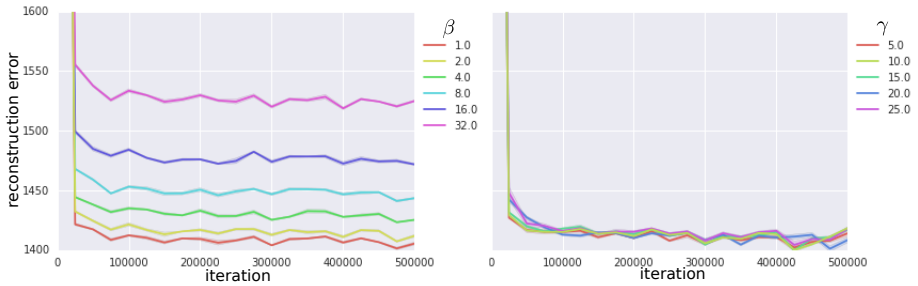


Figure 5.12: Plots of reconstruction error of β-VAE (left) and FactorVAE (right) for different values of β and γ on 3D Faces data over 5 random seeds.

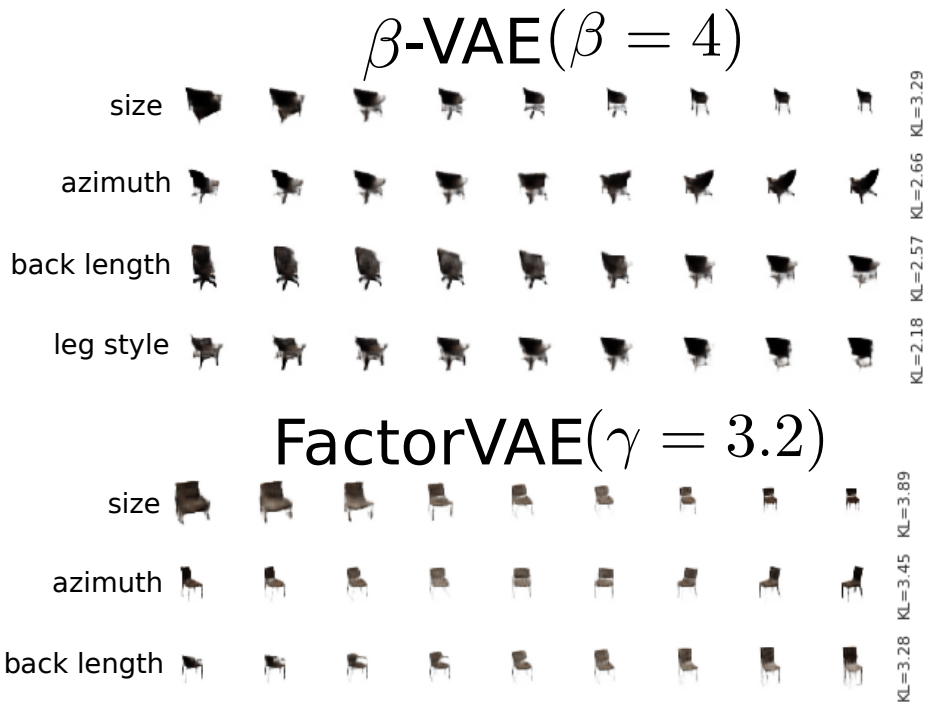


Figure 5.13: β-VAE and FactorVAE latent traversals across each latent dimension sorted by KL on 3D Chairs, with annotations of the factor of variation corresponding to each latent unit.

together with samples from the generative model in Appendix 5.9.

We also show results for β-VAE and FactorVAE experiments on the data sets with unknown generative factors, namely 3D Chairs, 3D Faces, and CelebA. Note that inspecting latent traversals is the only evaluation method possible here. We can see from Figure 5.12 (and Figures 5.38 and 5.39 in Appendix 5.9.4) that FactorVAE has smaller reconstruction error compared to β-VAE, and is capable of learning sensible factors of variation, as shown in the latent traversals in Figures 5.13, 5.14 and 5.15. Unfortunately, as explained in Section 5.4, latent traversals tell us little about the

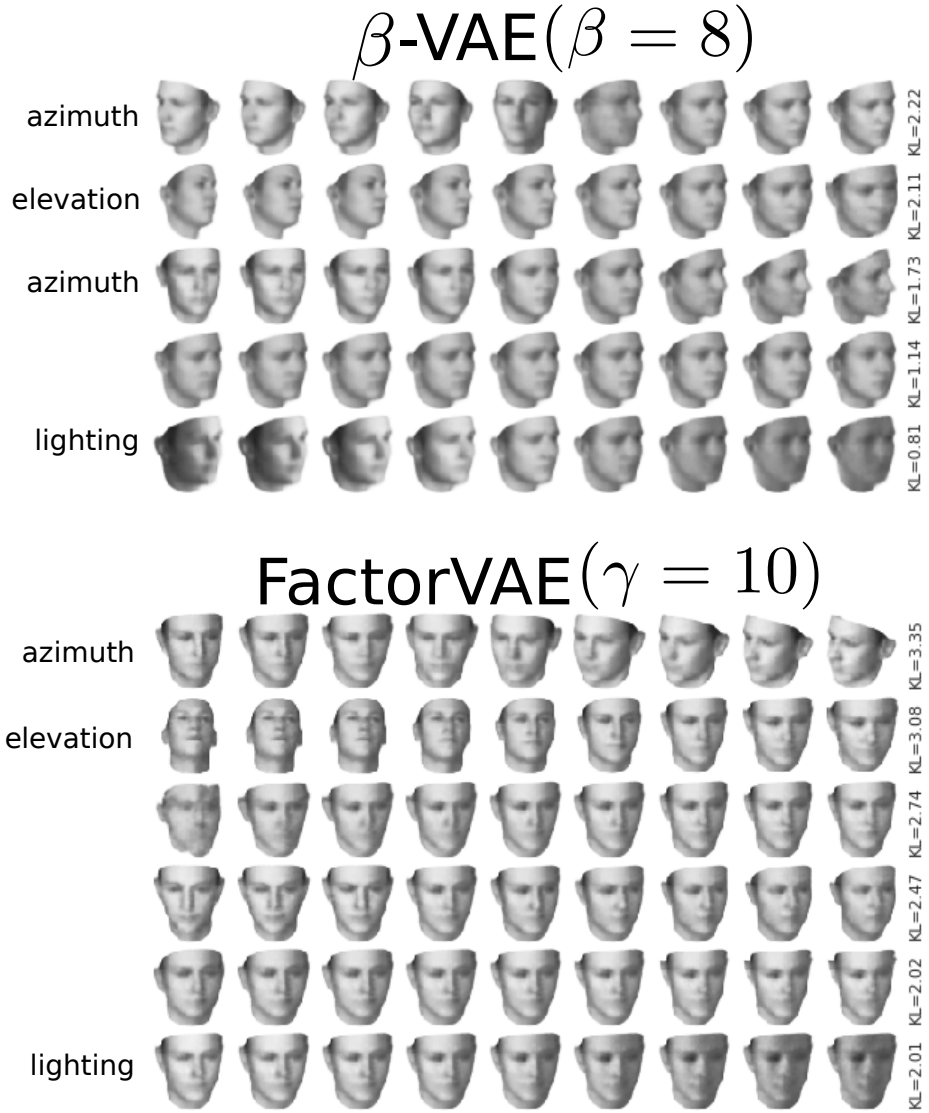


Figure 5.14: Same as Figure 5.13 but for 3D Faces.

robustness of our method.

5.7 Conclusion and Discussion

We have introduced FactorVAE, a novel method for disentangling that achieves better disentanglement scores than β-VAE on the 2D Shapes and 3D Shapes data sets for the same reconstruction quality. Moreover, we have identified weaknesses of the commonly used disentanglement metric of Higgins et al. [2016], and proposed an alternative metric that is conceptually simpler, is free of hyperparameters, and avoids the failure mode of the former. Finally, we have performed an experimental evaluation

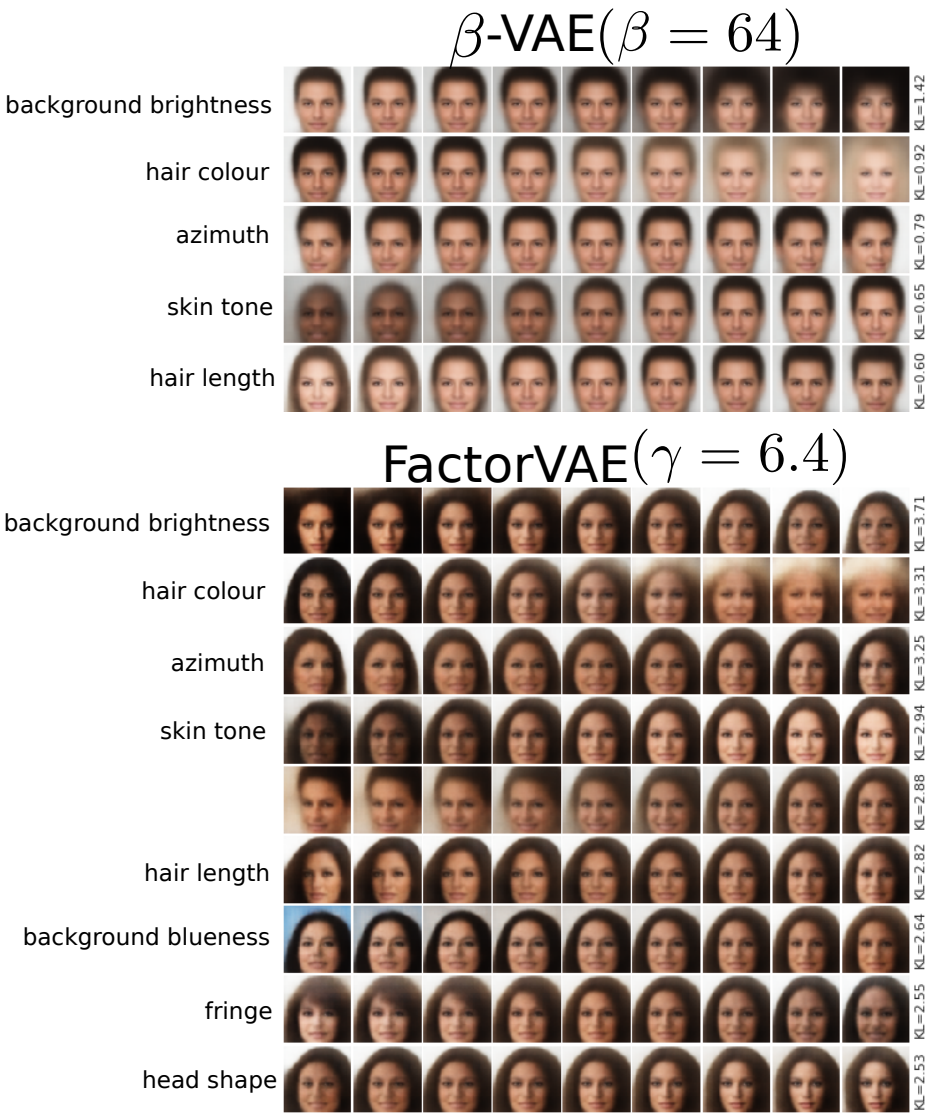


Figure 5.15: Same as Figure 5.13 but for CelebA.

of disentangling for the VAE-based methods and InfoWGAN-GP, a more stable variant of InfoGAN, and identified its weaknesses relative to the VAE-based methods.

One of the limitations of our approach is that low Total Correlation is necessary but not sufficient for disentangling of independent factors of variation. For example, if all but one of the latent dimensions were to collapse to the prior, the TC would be 0 but the representation would not be disentangled. Our disentanglement metric also requires us to be able to generate samples holding one factor fixed, which may not always be possible, for example when our training set does not cover all possible combinations of factors. The metric is also unsuitable for data with non-independent factors of variation.

For future work, we would like to use discrete latent variables to model discrete factors of variation and investigate how to reliably capture combinations of discrete and continuous factors using discrete and continuous latents.

5.7.1 Post-hoc Discussion

Ever since the publication of this paper, there has been a rapid growth of research on disentangling. Some advances were directly influenced by the content of this paper, whereas others were independent threads of research. Chen et al. [2018] have proposed alternative Monte Carlo estimates of the TC term in the FactorVAE loss; described as one of the challenges for disentangling in the paper, the task of disentangling discrete factors from continuous factors has been tackled in Dupont [2018]; the task of learning structured representations in an unsupervised manner and when the training set does not cover all combinations of factors, another set of challenges and limitations that were discussed, was explored in Esmaeili et al. [2018]. There have also been extensive empirical studies of various disentangling methods and metrics in Locatello et al. [2018], among which FactorVAE and the proposed metric appear as key baselines, with a thorough discussion of its properties and limitations. FactorVAE and the proposed metric also appear in `disentanglement_lib`[6], the largest open source library of models, metrics and data sets for disentangling as of today.

Locatello et al. [2018] also contains a theoretical result highlighting the identifiability issue in likelihood-based disentangling models: given such a model that has learned disentangled representations, there exists another model with the same likelihood that has entangled representations. Hence the paper claims that the role of inductive biases of the model structure and the data are key for disentangling rather than the loss. However the limitation of this argument is that the loss used for disentangling in VAE models, such as beta-VAE and FactorVAE, is not the marginal likelihood but variants of the ELBO, and the variational posterior is usually of restricted form, i.e. a factorised Gaussian. In fact, Rolinek et al. [2019] give theoretical explanations of how the ELBO, together with the factorised variational posterior, allows for the representations of a VAE to align with the local principal component directions of the data, avoiding the identifiability issue to some extent.

[6]Link: `https://github.com/google-research/disentanglement_lib`

Acknowledgements

We thank Chris Burgess and Nick Watters for providing the data sets and helping to set them up, and thank Guillaume Desjardins, Sergey Bartunov, Mihaela Rosca, Irina Higgins and Yee Whye Teh for helpful discussions.

5.8 Supplementary Material

5.8.1 Experimental Details for FactorVAE and β-VAE

We use a Convolutional Neural Network for the encoder, a Deconvolutional Neural Network for the decoder and a Multi-Layer Perceptron (MLP) with for the discriminator in FactorVAE for experiments on all data sets. We use [0,1] normalised data as targets for the mean of a Bernoulli distribution, using negative cross-entropy for $\log p(x|z)$ and Adam optimiser [Kingma and Ba, 2015] with learning rate 10^{-4}, $\beta_1 = 0.9, \beta_2 = 0.999$ for the VAE updates, as in Higgins et al. [2016]. We also use Adam for the discriminator updates with $\beta_1 = 0.5, \beta_2 = 0.9$ and a learning rate tuned from $\{10^{-4}, 10^{-5}\}$. We use 10^{-4} for 2D Shapes and 3D Faces, and 10^{-5} for 3D Shapes, 3D Chairs and CelebA. The encoder outputs parameters for the mean and log-variance of Gaussian $q(z|x)$, and the decoder outputs logits for each entry of the image. We use the same encoder/decoder architecture for β-VAE and FactorVAE, shown in Tables 5.1, 5.2, and 5.3. We use the same 6 layer MLP discriminator with 1000 hidden units per layer and leaky ReLU (lReLU) non-linearity, that outputs 2 logits in all FactorVAE experiments. We noticed that smaller discriminator architectures work fine, but noticed small improvements up to 6 hidden layers and 1000 hidden units per layer. Note that scaling the discriminator learning rate is not equivalent to scaling γ, since γ does not affect the discriminator loss. See Algorithm 4 for details of FactorVAE updates. We train for 3×10^5 iterations on 2D Shapes, 5×10^5 iterations on 3D Shapes, and 10^6 iterations on Chairs, 3D Faces and CelebA. We use a batch size of 64 for all data sets.

5.8.2 Details for the Disentanglement Metrics

We performed a sensitivity analysis of each metric with respect to its hyperparameters (c.f. Figure 5.2). In Figures 5.16, we show that the metric in Higgins et al. [2016] is very sensitive to number of iterations of the Adagrad [Duchi et al., 2011] optimiser

Table 5.1: Encoder and Decoder architecture for 2D Shapes data.

Encoder	**Decoder**
Input 64×64 binary image	Input $\in \mathbb{R}^{10}$
4×4 conv. 32 ReLU. stride 2	FC. 128 ReLU.
4×4 conv. 32 ReLU. stride 2	FC. $4 \times 4 \times 64$ ReLU.
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 64 ReLU. stride 2
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 32 ReLU. stride 2
FC. 128. FC. 2×10.	4×4 upconv. 32 ReLU. stride 2
	4×4 upconv. 1. stride 2

Table 5.2: Encoder and Decoder architecture for 3D Shapes, CelebA, Chairs data.

Encoder	**Decoder**
Input $64 \times 64 \times 3$ RGB image	Input $\in \mathbb{R}^{10}$
4×4 conv. 32 ReLU. stride 2	FC. 256 ReLU.
4×4 conv. 32 ReLU. stride 2	FC. $4 \times 4 \times 64$ ReLU.
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 64 ReLU. stride 2
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 32 ReLU. stride 2
FC. 256. FC. 2×10.	4×4 upconv. 32 ReLU. stride 2
	4×4 upconv. 3. stride 2

Table 5.3: Encoder and Decoder architecture for 3D Faces data.

Encoder	**Decoder**
Input 64×64 greyscale image	Input $\in \mathbb{R}^{10}$
4×4 conv. 32 ReLU. stride 2	FC. 256 ReLU.
4×4 conv. 32 ReLU. stride 2	FC. $4 \times 4 \times 64$ ReLU.
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 64 ReLU. stride 2
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 32 ReLU. stride 2
FC. 256. FC. 2×10.	4×4 upconv. 32 ReLU. stride 2
	4×4 upconv. 1. stride 2

with learning rate 0.01 (used in Higgins et al. [2016]), and constantly improves with more iterations. This suggests that one might want to use less noisy multi-class logistic regression solvers than gradient-descent based methods. The number of data points used to evaluate the metric after optimisation did not seem to help reduce variance beyond 800. So in our experiments, we use $L = 200$ and 10000 iterations, with a batch size of 10 per iteration of training the linear classifier, and use a batch of size 800 to evaluate the metric at the end of training. Each evaluation of this metric took around 30 minutes on a single GPU, hence we could not afford to train for more iterations.

For our disentanglement metric, we first prune out all latent dimensions that have

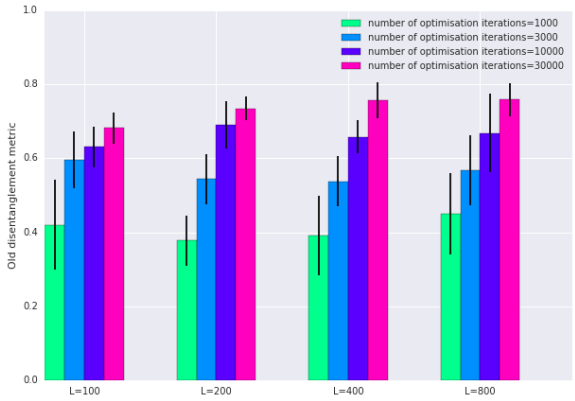


Figure 5.16: Mean and standard deviation of metric in Higgins et al. [2016] across 10 random seeds for varying L and number of Adagrad optimiser iterations (batch size 10). The number of points used for evaluation after optimisation is fixed to 800. These were all evaluated on a fixed, randomly chosen β-VAE model that was trained to convergence on the 2D Shapes data.

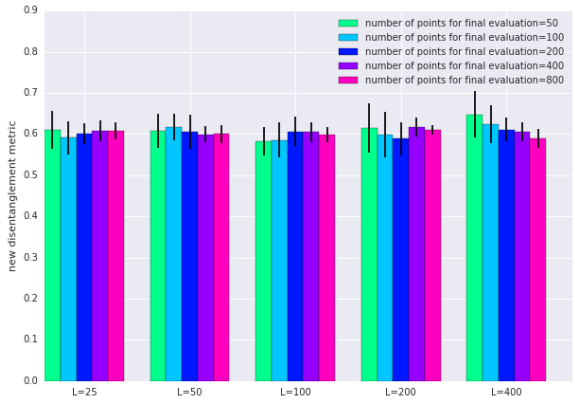


Figure 5.17: Mean and standard deviation of our metric across 10 random seeds for varying L and number of points used for evaluation. These were all evaluated on a fixed, randomly chosen β-VAE model that was trained to convergence on the 2D Shapes data.

collapsed to the prior ($q(z_j|x) = p(z_j)$). Then we just use the surviving dimensions for the majority vote. From the sensitivity analysis our metric in Figure 5.17, we observe that our metric is much less sensitive to hyperparameters than the metric in Higgins et al. [2016]. We use $L = 100$ and take the majority vote classifier from 800 votes. This only takes a few seconds on a single GPU. The majority vote classifier C works as follows: suppose we are given data $(a_i, b_i)_{i=1}^M, a_i \in \{1, \ldots, D\}, b_i \in \{1, \ldots, K\}$ (so $M = 500$). Then for $j \in \{1, \ldots, D\}$, let $V_{jk} = \sum_{i=1}^M \mathbb{I}(a_i = j, b_i = k)$. Then the majority vote classifier is defined to be $C(j) = \arg\max_k V_{jk}$.

Note that D, the dimensionality of the latents, does not affect the metric; for a

classifier that chooses at random, the accuracy is $1/K$, independent of D.

For discrete latent variables, we use Gini's definition of empirical variance:

$$\widehat{Var}(x) = \frac{1}{2N(N-1)} \sum_{i,j=1}^{N} d(x_i, x_j) \tag{5.4}$$

for $x = [x_1, \ldots, x_N] \in \mathbb{R}^N$, $d(x_i, x_j) = 1$ if $x_i \neq x_j$ and 0 if $x_i = x_j$. Note that this is equal to empirical variance for continuous variables when $d(x_i, x_j) = (x_i - x_j)^2$.

5.8.3 KL Decomposition

The KL term in the VAE objective decomposes as follows [Makhzani and Frey, 2017]:

Lemma 7. $\mathbb{E}_{p_{data}(x)}[KL(q(z|x)||p(z))] = I_q(x;z) + KL(q(z)||p(z))$ *where* $q(x,z) = p_{data}(x)q(z|x)$.

Proof.

$$\begin{aligned}
\mathbb{E}_{p_{data}(x)}[KL(q(z|x)||p(z))] &= \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{p(z)}\right] \\
&= \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{q(z)}\frac{q(z)}{p(z)}\right] \\
&= \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{q(z)} + \log \frac{q(z)}{p(z)}\right] \\
&= \mathbb{E}_{p_{data}(x)}[KL(q(z|x)||q(z))] + \mathbb{E}_{q(x,z)}\left[\log \frac{q(z)}{p(z)}\right] \\
&= I_q(x;z) + \mathbb{E}_{q(z)}\left[\log \frac{q(z)}{p(z)}\right] \\
&= I_q(x;z) + KL(q(z)||p(z))
\end{aligned}$$

□

Remark. *Note that this decomposition is equivalent to that in Hoffman and Johnson [2016], written as follows:* $\mathbb{E}_{p_{data}(x)}[KL(q(z|x)||p(z))] = I_r(i;z) + KL(q(z)||p(z))$ *where* $r(i,z) = \frac{1}{N}q(z|x^{(i)})$, *hence* $r(z|i) = q(z|x^{(i)})$, $r(z) = \frac{1}{N}\sum_{i=1}^{N} q(z|x^{(i)}) = q(z)$.

Proof.

$$\begin{aligned} I_r(i;z) &= \mathbb{E}_{r(i)}[KL(r(z|i)||r(z))] \\ &= \frac{1}{N}\sum_{i=1}^{N} KL(q(z|x^{(i)})||q(z)) \\ &= \mathbb{E}_{p_{data}(x)}[KL(q(z|x)||q(z))] \\ &= I_q(x;z) \end{aligned}$$

□

5.8.4 Using a Batch Estimate of $q(z)$ for Estimating TC

We have also tried using a batch estimate for the density $q(z)$, thus optimising this estimate of the TC directly instead of having a discriminator and using the density ratio trick. In other words, we tried $q(z) \approx \hat{q}(z) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} q(z|x^{(i)})$, and using the estimate:

$$KL(q(z)||\prod_j q(z_j)) = \mathbb{E}_{q(z)}\left[\log \frac{q(z)}{\prod_j q(z_j)}\right] \approx \mathbb{E}_{q(z)}\left[\log \frac{\hat{q}(z)}{\prod_j \hat{q}(z_j)}\right] \tag{5.5}$$

Note that:

$$\mathbb{E}_{q(z)}\left[\log \frac{\hat{q}(z)}{\prod_j \hat{q}(z_j)}\right] \approx \frac{1}{H}\sum_{h=1}^{H}\left[\log \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\prod_{j=1}^{D} q(z_j^{(h)}|x^{(i)}) - \log \prod_{j=1}^{D}\frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} q(z_j^{(h)}|x^{(i)})\right] \tag{5.6}$$

for $z^{(h)} \overset{iid}{\sim} q(z)$. However while experimenting on 2D Shapes, we observed that the value of $\log q(z^{(h)})$ becomes very small (negative with high absolute value) for latent dimension $d \geq 2$ during training, because $\hat{q}(z)$ is not a good enough approximation to $q(z)$ unless $\mathcal{B}$ is very big. As training progresses for the VAE, the variance of Gaussians $q(z|x^{(i)})$ becomes smaller and smaller, so they do not overlap too much in higher dimensions. Hence we get $z^{(h)} \sim q(z)$ that land on the tails of $\hat{q}(z) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} q(z|x^{(i)})$, giving worryingly small values of $\log \hat{q}(z^{(h)})$. On the other hand $\prod_j \hat{q}(z_j^{(h)})$, a mixture of $|\mathcal{B}|^d$ Gaussians hence of much higher entropy, gives much more stable values of $\log \prod_j \hat{q}(z_j^{(h)})$. From Figure 5.18, we can see that even with $\mathcal{B}$ as big as 10,000, we get negative values for the estimate of TC, which is a KL divergence and hence should be non-negative, hence this method of using a batch estimate for $q(z)$ does not work. A fix is to use samples from $\hat{q}(z)$ instead of $q(z)$, but this seemed to give a similar reconstruction-disentanglement trade-off to β-VAE. Very recently, work from Chen et al. [2018] has shown that disentangling can be improved by using samples from $\hat{q}(z)$.

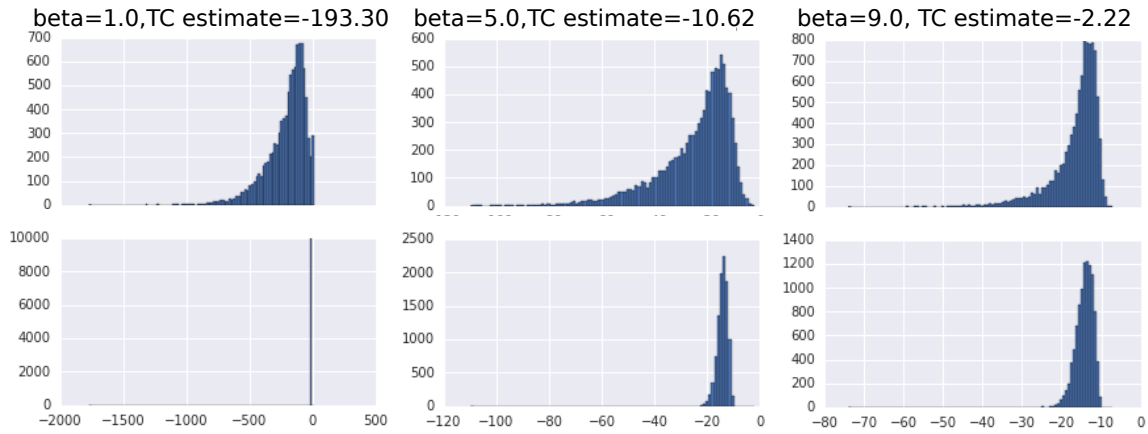


Figure 5.18: Histogram of $\log \hat{q}(z^{(h)})$ (top) and $\prod_{j=1}^{d} \hat{q}(z_j^{(h)})$ (bottom) for $z^{(h)} \overset{iid}{\sim} q(z)$ with $|\mathcal{B}| = 10000$, $d = 10$. The columns correspond to values of $\beta = 1, 5, 9$ for training β-VAE. In the title of each histogram, there is an estimate of TC based on the samples of $z^{(h)}$.

5.9 Log Marginal Likelihood and Samples

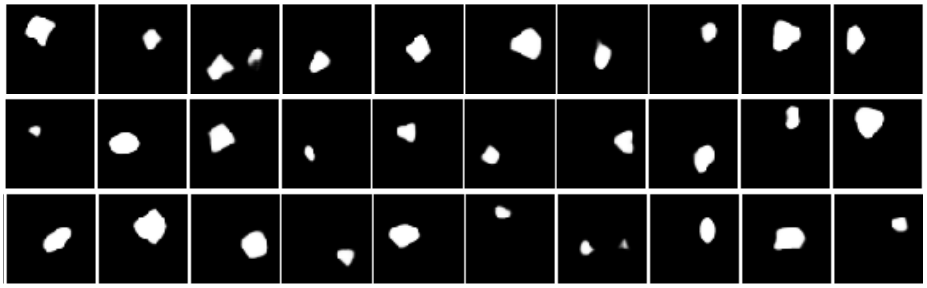


Figure 5.19: Randomly chosen samples from the best performing (in terms of disentanglement) β-VAE generative model ($\beta = 4$).

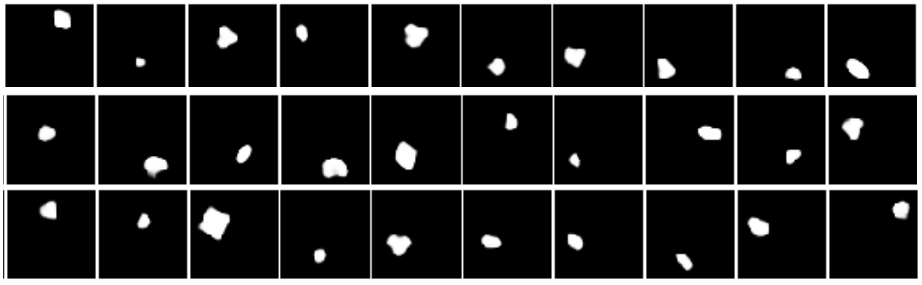


Figure 5.20: Randomly chosen samples from the best performing (in terms of disentanglement) FactorVAE generative model ($\gamma = 35$).

We give the log marginal likelihood of each of the best performing β-VAE and FactorVAE models (in terms of disentanglement) for both the 2D Shapes and 3D

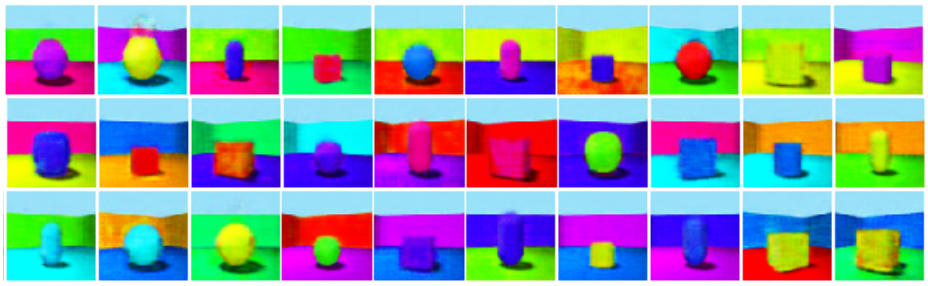


Figure 5.21: Randomly chosen samples from the best performing (in terms of disentanglement) β-VAE generative model ($\beta = 32$).

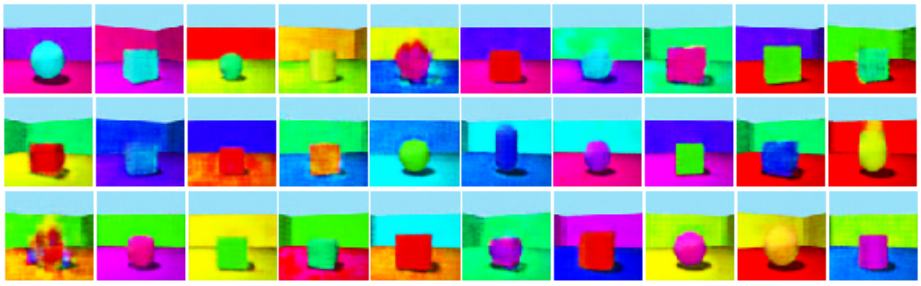


Figure 5.22: Randomly chosen samples from the best performing (in terms of disentanglement) FactorVAE generative model ($\gamma = 6$).

Shapes data sets along with samples from the generative model. Since the log marginal likelihood is intractable, we report the *Importance-Weighted Autoencoder* (IWAE) bound with 5000 particles, in line with standard practice in the generative modelling literature.

In Figures 5.19 and 5.20, the samples for FactorVAE are arguably more representative of the data set than those of β-VAE. For example β-VAE has occasional samples with two separate shapes in the same image (Figure 5.19). The log marginal likelihood for the best performing β-VAE ($\beta = 4$) is -46.1, whereas for FactorVAE it is -51.9 ($\gamma = 35$) (a randomly chosen VAE run gives -43.3). So on 2D Shapes, FactorVAE gives better samples but worse log marginal likelihood.

In Figures 5.21 and 5.22, the samples for β-VAE appear more coherent than those for FactorVAE. However the log marginal likelihood for β-VAE ($\beta = 32$) is -3534, whereas for FactorVAE it is -3520 ($\gamma = 6$) (a randomly chosen VAE run gives -3517). So on 3D Shapes, FactorVAE gives worse samples but better log marginal likelihood.

In general, if one seeks to learn a generative model with a disentangled latent space, it would make sense to choose the model with the lowest value of β or γ among those with similarly high disentanglement performance.

5.9.1 Losses and Experiments for other related Methods

The Adversarial Autoencoder (AAE) [Makhzani et al., 2015] uses the following objective

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)]\Big] - KL(q(z)||p(z)), \tag{5.7}$$

utilising the density ratio trick to estimate the KL term.

Information Dropout [Achille and Soatto, 2018] uses the objective

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - \beta KL(q(z|x^{(i)})||q(z)). \tag{5.8}$$

The following objective is also considered in the paper but is dismissed as intractable:

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - \beta KL(q(z|x^{(i)})||q(z))\Big] - \gamma KL(q(z)||\prod_{j=1}^{d}q(z_j)) \tag{5.9}$$

Note that it is similar to the FactorVAE objective (which has $\beta = 1$), but with $p(z)$ in the first KL term replaced with $q(z)$.

DIP-VAE [Kumar et al., 2018a] uses the VAE objective with an additional penalty on how much the covariance of $q(z)$ deviates from the identity matrix, either using the law of total covariance $Cov_{q(z)}[z] = \mathbb{E}_{p_{data}(x)}Cov_{q(z|x)}[z] + Cov_{p_{data}(x)}(\mathbb{E}_{q(z|x)}[z])$ (DIP-VAE I):

$$\begin{aligned}&\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - KL(q(z|x^{(i)})||p(z))\Big]\\ &-\lambda_{od}\sum_{i\neq j}[Cov_{p_{data}(x)}[\mu(x)]]_{ij}^2 - \lambda_d\sum_i([Cov_{p_{data}(x)}[\mu(x)]]_{ii}-1)^2\end{aligned} \tag{5.10}$$

where $\mu(x) = mean(q(z|x))$, or directly (DIP-VAE II):

$$\begin{aligned}&\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - KL(q(z|x^{(i)})||p(z))\Big]\\ &-\lambda_{od}\sum_{i\neq j}[Cov_{q(z)}[z]]_{ij}^2 - \lambda_d\sum_i([Cov_{q(z)}[z]]_{ii}-1)^2\end{aligned} \tag{5.11}$$

One could argue that during training of FactorVAE, $\prod_j q(z_j)$ will be similar to $p(z)$, assuming the prior is factorial, due to the $KL(q(z|x)||p(z))$ term in the objective.

Hence we also investigate a modified FactorVAE objective that replaces $\prod_j q(z_j)$ with $p(z)$:

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - KL(q(z|x^{(i)})||p(z))\Big] - \gamma KL(q(z)||p(z)) \tag{5.12}$$

However as shown in Figure 5.40 of Appendix 5.9.4, the histograms of samples from the marginals are clearly quite different from the prior for FactorVAE.

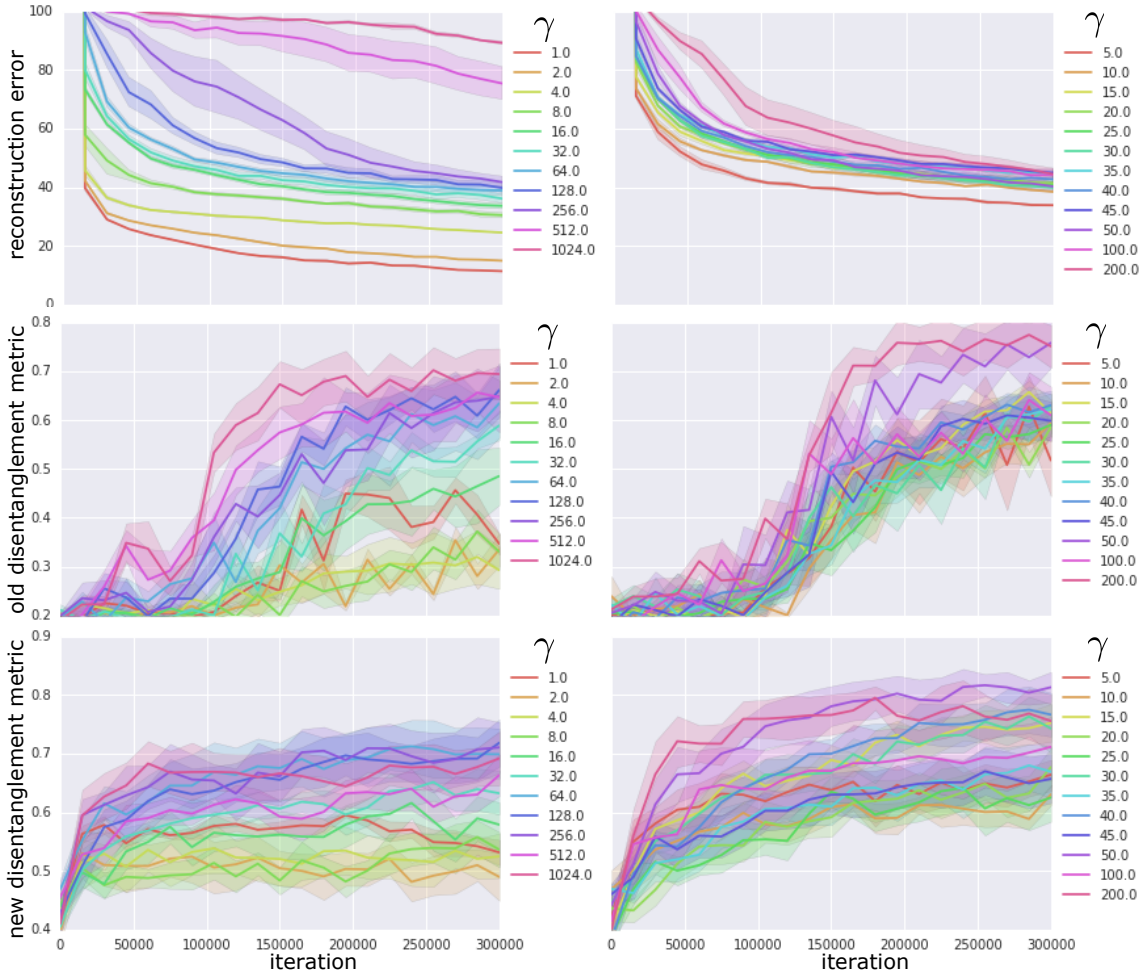


Figure 5.23: Same as Figure 5.4 but for AAE (left) and a variant of FactorVAE (Equation (5.12).

Moreover we show experimental results for AAE (adding a γ coefficient in front of the $KL(q(z)||p(z))$ term of the objective and tuning it) and the variant of FactorVAE (Equation (5.12)) on the 2D Shapes data. From Figure 5.23, we see that the disentanglement performance for both are somewhat lower than that for FactorVAE. This difference could be explained as a benefit of directly encouraging $q(z)$ to be factorised

(FactorVAE) instead of encouraging it to approach an arbitrarily chosen factorised prior $p(z) = \mathcal{N}(0, I)$ (AAE, Equation (5.12)). Information Dropout and DIP-VAE did not have enough experimental details in the paper nor publicly available code to have their results reproduced and compared against.

5.9.2 InfoGAN and InfoWGAN-GP

We give an overview of InfoGAN [Chen et al., 2016] and InfoWGAN-GP, its counterpart using Wasserstein distance and gradient penalty. InfoGAN uses latents $z = (c, \epsilon)$ where c models semantically meaningful codes and ϵ models incompressible noise. The generative model is defined by a generator G with the process: $c \sim p(c), \epsilon \sim p(\epsilon), z = (c, \epsilon)$, $x = G(z)$. i.e. $p(z) = p(c)p(\epsilon)$. GANs are defined as a minimax game on some objective $V(D, G)$, where D is either a discriminator (e.g. for the original GAN [Goodfellow et al., 2014a]) that outputs log probabilities for binary classification, or a critic (e.g. for Wasserstein-GAN [Arjovsky et al., 2017]) that outputs a real-valued scalar. InfoGAN defines an extra encoding distribution $Q(c|x)$ that is used to define an extra penalty:

$$L(G, Q) = \mathbb{E}_{p(c)}\mathbb{E}_{p(\epsilon)}[\log Q(c|G(c, \epsilon))] \tag{5.13}$$

that is added to the GAN objective. Hence InfoGAN is the following minimax game on the parameters of neural nets D, G, Q:

$$\min_{G,Q} \max_{D} V_I(D, G, Q) = \min_{G,Q} \max_{D} V(D, G) - \lambda L(G, Q) \tag{5.14}$$

L can be interpreted as a variational lower bound to $I(c; G(c, \epsilon))$, with equality at $Q = \arg\min_Q V_I(D, G, Q)$. i.e. L encourages the codes to be more informative about the image. From the definition of L, it can also be seen as the reconstruction error of codes in the latent space. The original InfoGAN defines:

$$V(D, G) = \mathbb{E}_{p_{data}(x)}[D(x)] - \mathbb{E}_{p(z)}[D(G(z))] \tag{5.15}$$

same as the original GAN objective where D outputs log probabilities. However as we'll show in Appendix 5.9.3 this has known instability issues in training. So it is natural to try replacing this with the more stable WGAN-GP [Gulrajani et al., 2017] objective:

$$\begin{aligned} V(D, G) =& \mathbb{E}_{p_{data}(x)}[D(x)] - \mathbb{E}_{p(z)}[D(G(z))] \\ &+ \eta(||\nabla_x D(x)|_{x=\hat{x}}||_2 - 1)^2 \end{aligned} \tag{5.16}$$

for $\hat{x} = \pi x_r + (1-\pi)x_f$ with $\pi \sim U[0, 1]$, $x_r \sim p_{data}(x)$, $z_f \sim p(z)$, $x_f = \texttt{stop\_gradient}(G(z_f))$ and with a new $\hat{x}$ for each iteration of optimisation. Thus we obtain InfoWGAN-GP.

5.9.3 Empirical Study of InfoGAN and InfoWGAN-GP

To begin with, we implemented InfoGAN and InfoWGAN-GP on MNIST using the hyperparameters given in Chen et al. [2016] to better understand its behaviour, using 1 categorical code with 10 categories, 2 continuous codes, and 62 noise variables. We use priors $p(c_j) = U[-1, 1]$ for the continuous codes, $p(c_j) = \frac{1}{J}$ for categorical codes with J categories, and $p(\epsilon_j) = \mathcal{N}(0, 1)$ for the noise variables. For 2D Shapes data we use 1 categorical codes with 3 categories ($J = 3$), 4 continuous codes, and 5 noise variables. The number of noise variables did not seem to have a noticeable effect on the experiment results. We use the Adam optimiser [Kingma and Ba, 2015] with $\beta_1 = 0.5, \beta_2 = 0.999$, and learning rate 10^{-3} for the generator updates and 10^{-4} for the discriminator updates. The detailed Discriminator/Encoder/Generator architecture are given in Tables 5.4 and 5.5. The architecture for InfoWGAN-GP is the same as InfoGAN, except that we use no Batch Normalisation (batchnorm) [Ioffe and Szegedy, 2015] for the convolutions in the discriminator, and replace batchnorm with Layer Normalisation [Ba et al., 2016] in the fully connected layer that follows the convolutions as recommended in [Gulrajani et al., 2017]. We use gradient penalty coefficient $\eta = 10$, again as recommended.

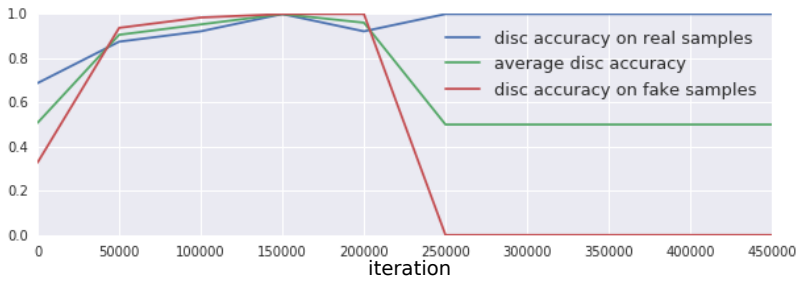


Figure 5.24: Discriminator accuracy of InfoGAN on MNIST throughout training.

We firstly observe that for all runs, we eventually get a degenerate discriminator that predicts all inputs to be real, as in Figure 5.24. This is the well-known instability issue of the original GAN. We have tried using a smaller learning rate for the discriminator, and although this delays the degenerate behaviour it does not prevent it. Hence early stopping seems crucial, and all results shown below are from well before the degenerate behaviour occurs.

Chen et al. [2016] claim that the categorical code learns digit class (discrete factor of variation) and that the continuous codes learn azimuth and width, but when plotting latent traversals for each run, we observed that this is inconsistent. We show five

Table 5.4: InfoGAN architecture for MNIST data. 2 continuous codes, 1 categorical code with 10 categories, 62 noise variables.

discriminator D / **encoder** Q	**generator** G
Input 28×28 greyscale image	Input $\in \mathbb{R}^{74}$
4×4 conv. 64 lReLU. stride 2	FC. 1024 ReLU. batchnorm
4×4 conv. 128 lReLU. stride 2. batchnorm	FC. $7 \times 7 \times 128$ ReLU. batchnorm
FC. 1024 lReLU. batchnorm	4×4 upconv. 64 ReLU. stride 2. batchnorm
FC. 1. output layer for D	4×4 upconv. 1 Sigmoid. stride 2
FC. 128 lReLU. batchnorm. FC $2 \times 2 + 1 \times 10$	

Table 5.5: InfoGAN architecture for 2D Shapes data. 4 continuous codes, 1 categorical code with 3 categories, 5 noise variables.

discriminator D / **encoder** Q	**generator** G
Input 64×64 binary image	Input $\in \mathbb{R}^{12}$
4×4 conv. 32 lReLU. stride 2	FC. 128 ReLU. batchnorm
4×4 conv. 32 lReLU. stride 2. batchnorm	FC. $4 \times 4 \times 64$ ReLU. batchnorm
4×4 conv. 64 lReLU. stride 2. batchnorm	4×4 upconv. 64 lReLU. stride 2. batchnorm
4×4 conv. 64 lReLU. stride 2. batchnorm	4×4 upconv. 32 lReLU. stride 2. batchnorm
FC. 128 lReLU. batchnorm	4×4 upconv. 32 lReLU. stride 2. batchnorm
FC. 1. output layer for D	4×4 upconv. 1 Sigmoid. stride 2
FC. 128 lReLU. batchnorm. FC $4 \times 2 + 1 \times 3$ for Q	

Table 5.6: Bigger InfoGAN architecture for 2D Shapes data. 4 continuous codes, 1 categorical code with 3 categories, 128 noise variables.

discriminator D / **encoder** Q	**generator** G
Input 64×64 binary image	Input $\in \mathbb{R}^{136}$
4×4 conv. 64 lReLU. stride 2	FC. 1024 ReLU. batchnorm
4×4 conv. 128 lReLU. stride 2. batchnorm	FC. $8 \times 8 \times 256$ ReLU. batchnorm
4×4 conv. 256 lReLU. stride 2. batchnorm	4×4 upconv. 256 lReLU. stride 1. batchnorm
4×4 conv. 256 lReLU. stride 1. batchnorm	4×4 upconv. 256 lReLU. stride 1. batchnorm
4×4 conv. 256 lReLU. stride 1. batchnorm	4×4 upconv. 128 lReLU. stride 2. batchnorm
FC. 1024 lReLU. batchnorm	4×4 upconv. 64 lReLU. stride 2. batchnorm
FC. 1. output layer for D	4×4 upconv. 1 Sigmoid. stride 2
FC. 128 lReLU. batchnorm. FC $4 \times 2 + 1 \times 3$ for Q	

randomly chosen runs in Figure 5.25. The digit class changes in the continuous code traversals and there are overlapping digits in the categorical code traversal. Similar results hold for InfoWGAN-GP in Figure 5.36.

We also tried visualising the reconstructions: given an image, we push the image

Figure 5.25: Latent traversals for InfoGAN on MNIST across the two continuous codes (first two rows) and the categorical code (last row) for 5 different random seeds.

through the encoder to obtain latent codes c, fix this c and vary the noise ϵ to generate multiple reconstructions for the same image. This is to check the extent to which the noise ϵ can affect the generation. We can see in Figure 5.26 that digit class often changes when varying ϵ, so the model struggles to cleanly separate semantically meaningful information and incompressible noise.

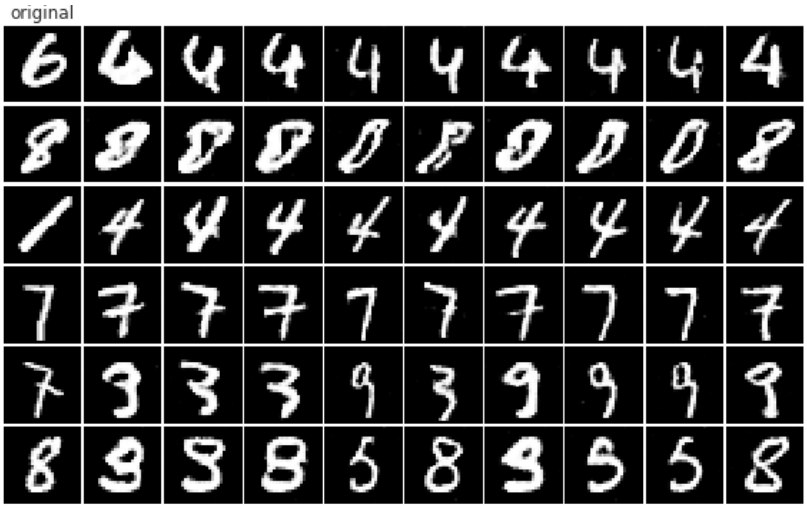


Figure 5.26: Reconstructions for InfoGAN on MNIST. First column: original image. Remaining columns: reconstructions varying the noise latent ϵ.

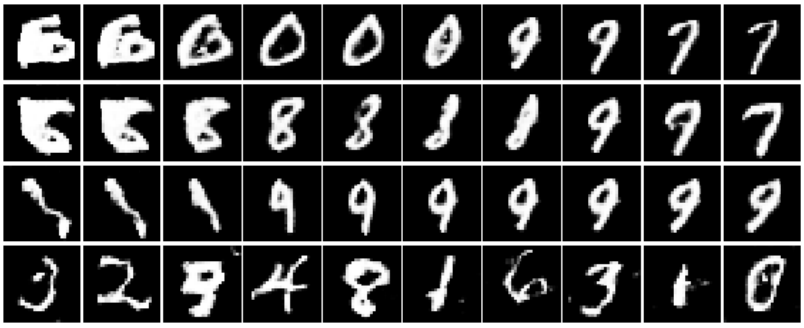


Figure 5.27: Latent traversals for InfoGAN on MNIST across the three continuous codes (first three rows) and the categorical code (last row).

Furthermore, we investigated the sensitivity of the model to the number of latent codes. We show latent traversals using three continuous codes instead of two in Figure 5.27. It is evident that the model tries to put more digit class information into the continuous traversals. So the number of codes is an important hyperparameter to tune, whereas VAE methods are less sensitive to the choice of number of codes since they can prune out unnecessary latents by collapsing $q(z_j|x)$ to the prior $p(z_j)$.

We also tried varying the number of categories for the categorical code. Using 2 categories, we see from Figure 5.28 that the model tries to put much more information about digit class into the continuous latents, as expected. Moreover from Figure 5.30, we can see that the noise variables also have more information about the digit class. However, when we use 20 categories, we see that the model still puts information about the digit class in the continuous latents. However from Figure 5.31 we see that

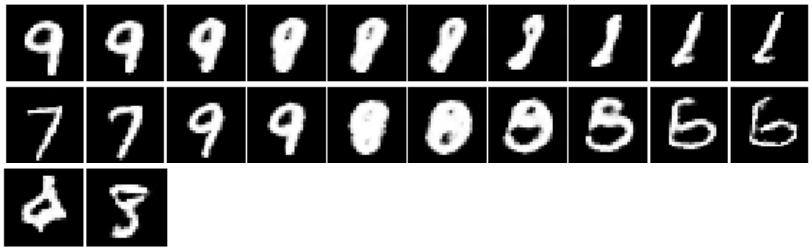


Figure 5.28: Latent traversals for InfoGAN on MNIST across the two continuous codes (first two rows) and the categorical code (last row) using 2 categories

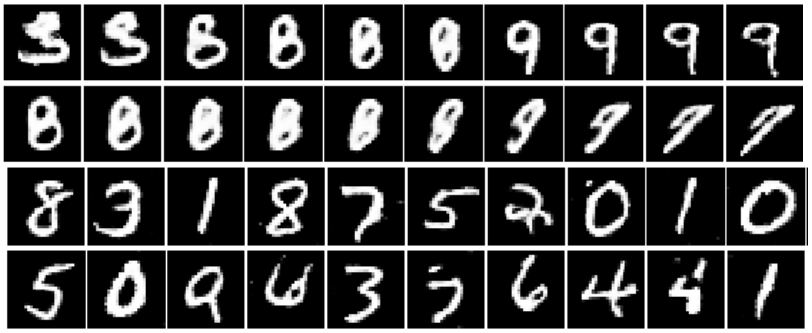


Figure 5.29: Latent traversals for InfoGAN on MNIST across the two continuous codes (first two rows) and the categorical code (last two rows) using 20 categories

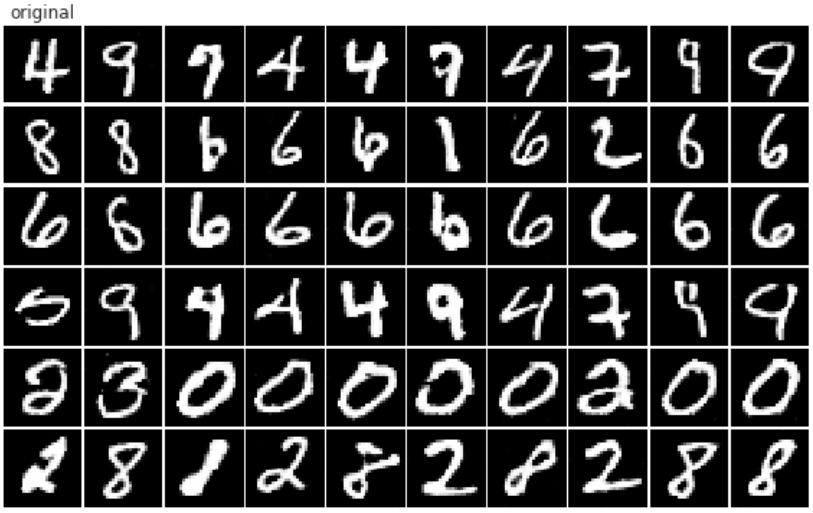


Figure 5.30: Same as Figure 5.26 but the categorical code having 2 categories.

the noise variables contain less semantically meaningful information.

Using InfoWGAN-GP solved the degeneracy issue and makes training more stable (see Figure 5.33), but we observed that the other problems persisted (see e.g. Figure 5.36).

For 2D Shapes, we have also tried using a bigger architecture for InfoWGAN-GP that is used for a data set of similar dimensions (Chairs data set) in Chen et al. [2016]. See

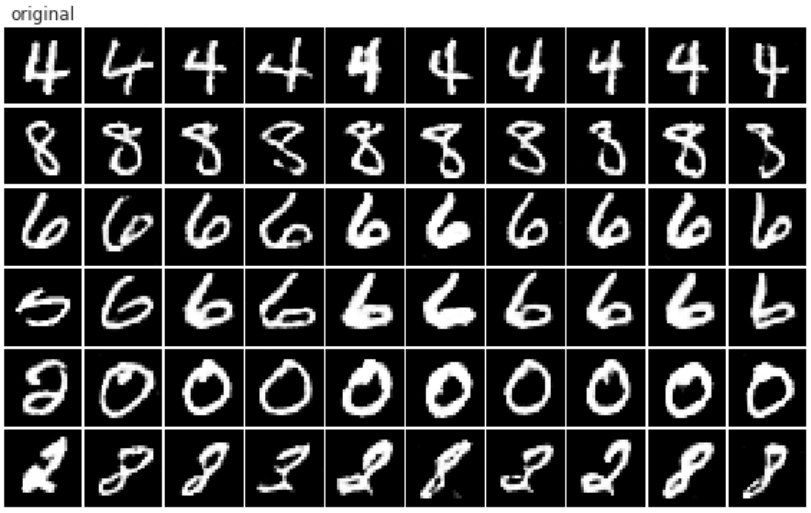


Figure 5.31: Same as Figure 5.30 but with 20 categories.

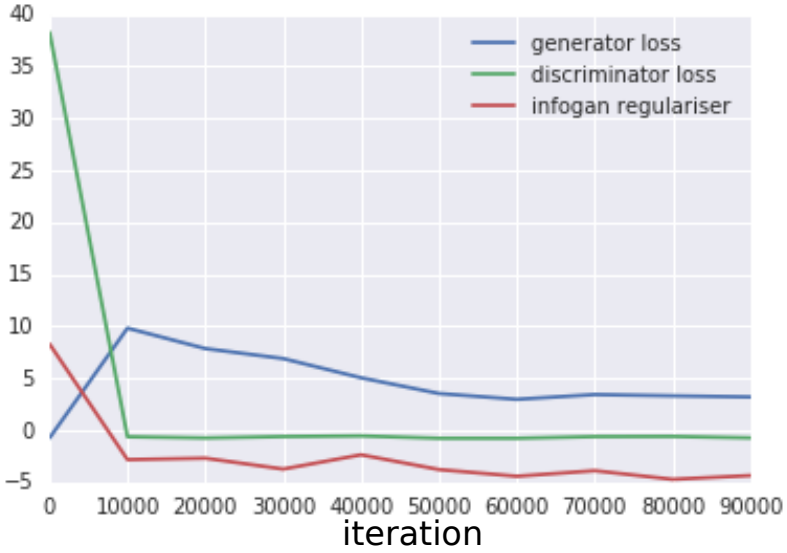


Figure 5.32: The generator loss $-\mathbb{E}_{p(z)}[D(G(z))]$, discriminator loss $\mathbb{E}_{p_{data}(x)}[D(x)] - \mathbb{E}_{p(z)}[D(G(z))]$ and the InfoGAN regulariser term $-L$ for InfoWGAN-GP on MNIST with $\lambda = 1$

Table 5.6. However as can be seen in Figure 5.34 this did not improve disentanglement scores, yet the latent traversals look slightly more realistic (Figure 5.35).

In summary, InfoWGAN-GP can help prevent the instabilities in training faced by InfoGAN, but it does not help overcome the following weaknesses compared to VAE-based methods: 1) Disentangling performance is sensitive to the number of code latents. 2) More often than not, the noise variables contain semantically meaningful information. 3) The model does not always generalise well to all across the domain of $p(z)$.

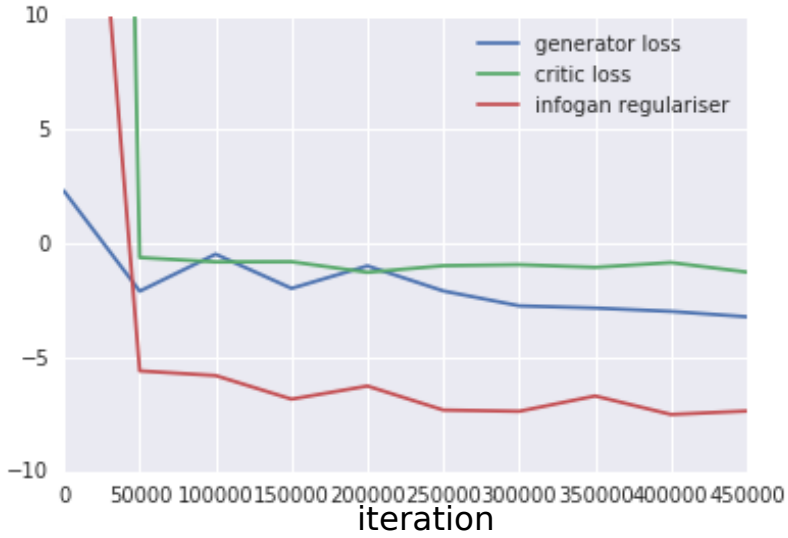


Figure 5.33: Same as Figure 5.32 but for 2D Shapes.

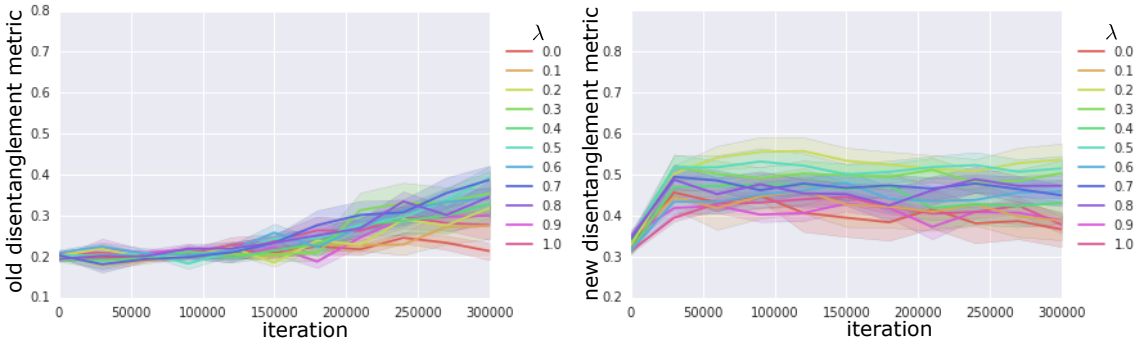


Figure 5.34: Disentanglement scores for InfoWGAN-GP on 2D Shapes with bigger architecture (Table 5.6) for 10 random seeds per hyperparameter setting. Left: Metric in Higgins et al. [2016]. Right: Our metric.

5.9.4 Further Experimental Results

From Figure 5.37, we see that higher values of γ in FactorVAE leads to a lower discriminator accuracy. This is as expected, since a higher γ encourages $q(z)$ and $\prod_j q(z_j)$ to be closer together, hence a lower accuracy for the discriminator to successfully classify samples from the two distributions.

We also show histograms of $q(z_j)$ for each j in β-VAE and FactorVAE for different values of β and γ at the end of training on 2D Shapes in Figure 5.40. We can see that the marginals of FactorVAE are quite different from the prior, which could be a reason that the variant of FactorVAE using the objective given by Equation (5.12) leads to different results to FactorVAE. For FactorVAE, the model is able to focus on

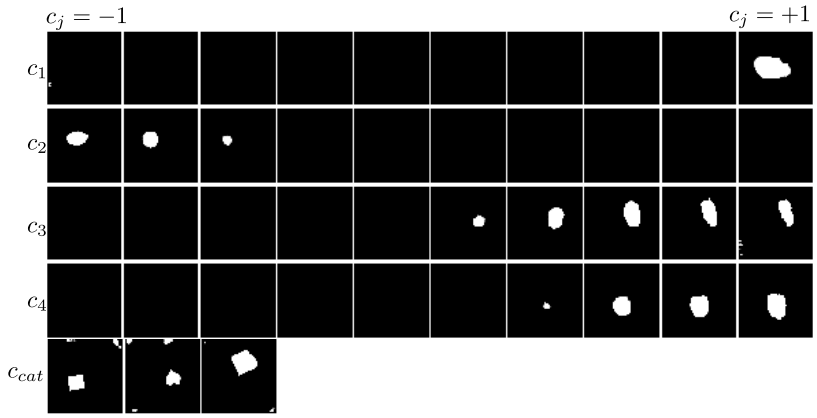


Figure 5.35: Latent traversals for InfoWGAN-GP on 2D Shapes across the four continuous codes (first four rows) and the categorical code (last row) with bigger architecture (Table 5.6) for run with best disentanglement score ($\lambda = 0.6$).

factorising $q(z)$ instead of pushing it towards some arbitrarily specified prior $p(z)$.

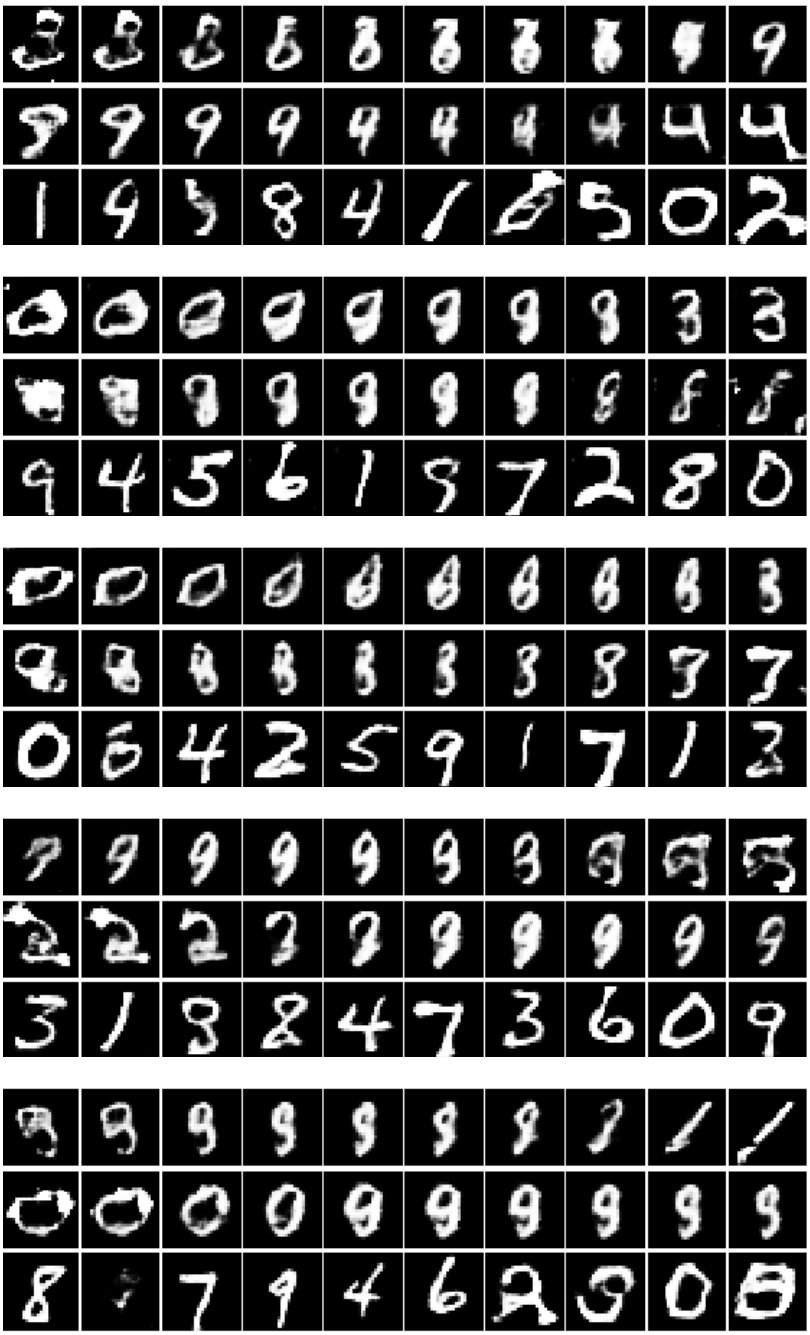


Figure 5.36: Same as Figure 5.25 but for InfoWGAN-GP.

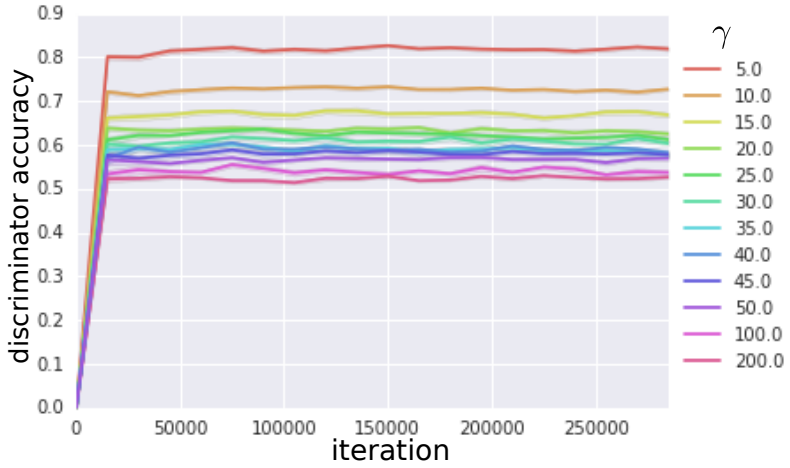


Figure 5.37: Plot of discriminator accuracy of FactorVAE on 2D Shapes data across iterations over 5 random seeds.

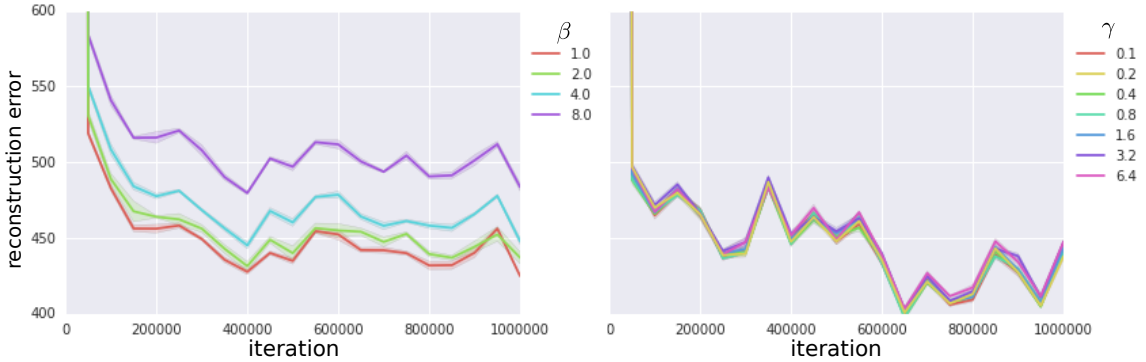


Figure 5.38: Same as Figure 5.12 but for 3D Chairs.

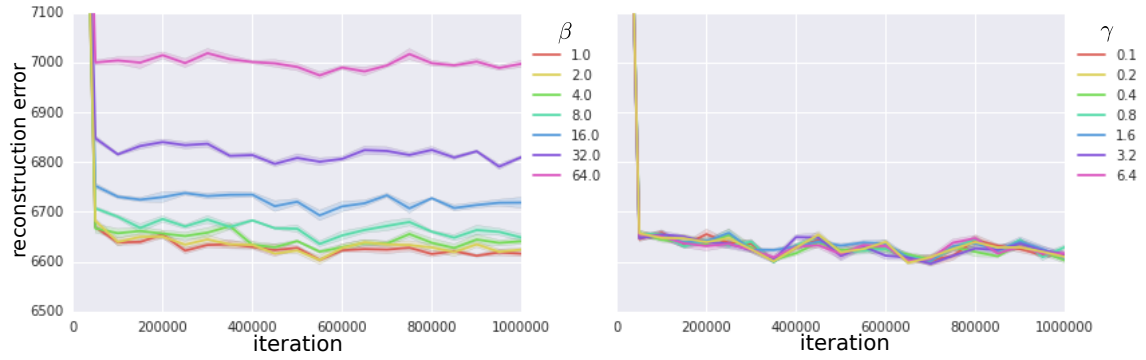


Figure 5.39: Same as Figure 5.12 but for CelebA.

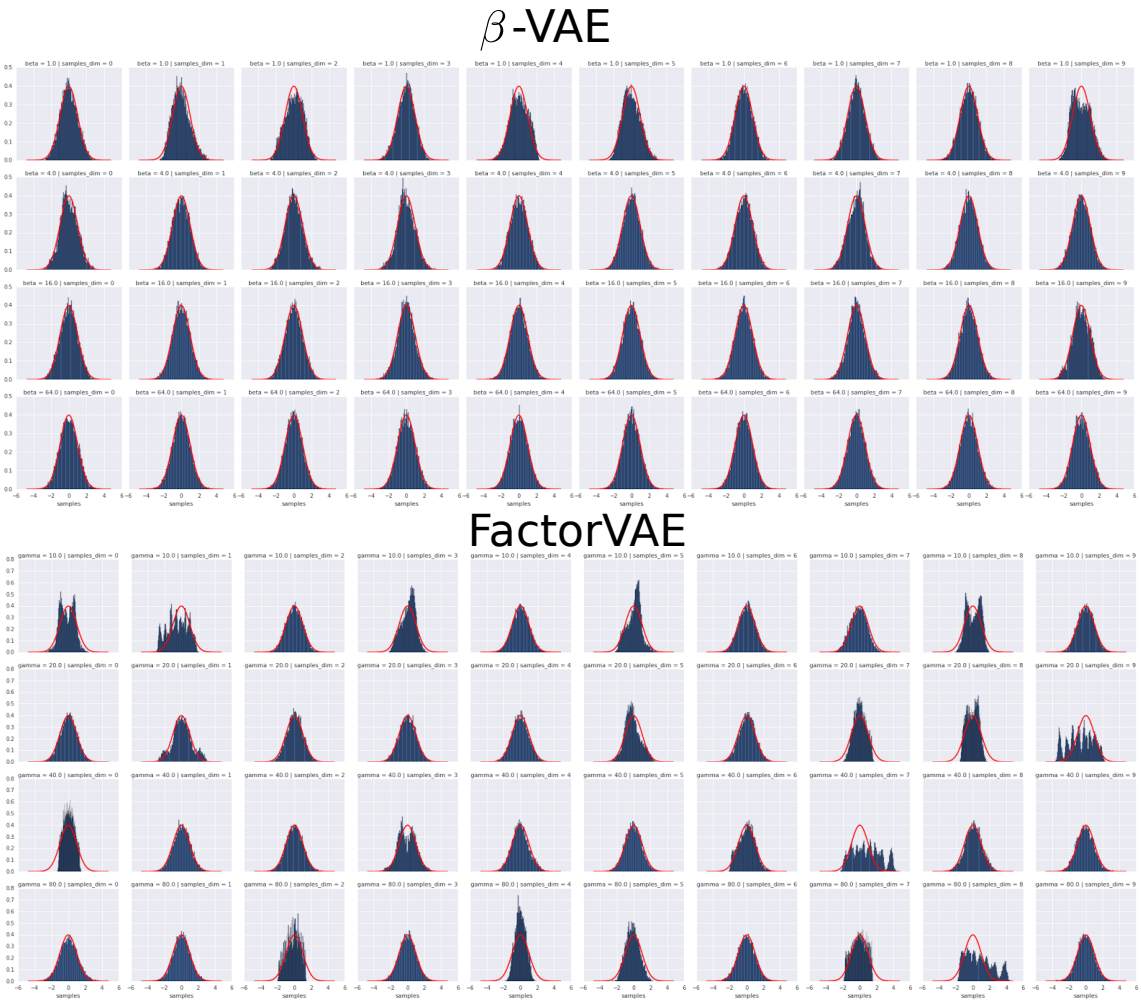


Figure 5.40: Histograms of $q(z_j)$ for each j (columns) for β-VAE and FactorVAE at the end of training on 2D Shapes, with the pdf of Gaussian $\mathcal{N}(0, 1)$ overlaid in red. The rows correspond to different values of β $(1, 4, 16, 64)$ and γ $(10, 20, 40, 80)$ respectively.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Disentangling by Factorising
Publication Status	x Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Kim, H. and Mnih, A., 2018, July. Disentangling by Factorising. In Proceedings of the 35th International Conference on Machine Learning (ICML).

Student Confirmation

Student Name:	Hyun Jik Kim		

Contribution to the Paper	- Solo lead of the project, with single advisor.
	- Formulated the problem that the paper tackles.
	- Came up with all central ideas for tackling the problem, including:
	  - Idea of disentangling by encouraging dimensions to be independent.
	  - Idea of approximating the Total Correlation term with a discriminator.
	  - Design of the new disentanglement metric.
	- Implemented the algorithm in code and carried out all experiments.
	- All of the paper writing (with feedback from advisors).		

Signature		Date	

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Yee Whye Teh, Professor of Statistical Machine Learning		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

Chapter 6

Attentive Neural Processes

The following chapter is a self-contained paper:

> **Kim, H.**, Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, M., Vinyals, O. and Teh, Y.W., 2019, May. Attentive Neural Processes. In Proceedings of the International Conference on Learning Representations (ICLR).

There is a statement of authorship at the end of this chapter.

Abstract

Neural Processes (NPs) [Garnelo et al., 2018a,b] approach regression by learning to map a context set of observed input-output pairs to a distribution over regression functions. Each function models the distribution of the output given an input, conditioned on the context. NPs have the benefit of fitting observed data efficiently with linear complexity in the number of context input-output pairs, and can learn a wide family of conditional distributions; they learn predictive distributions conditioned on context sets of arbitrary size. Nonetheless, we show that NPs suffer a fundamental drawback of underfitting, giving inaccurate predictions at the inputs of the observed data they condition on. We address this issue by incorporating attention into NPs, allowing each input location to attend to the relevant context points for the prediction. We show that this greatly improves the accuracy of predictions, results in noticeably faster training, and expands the range of functions that can be modelled.

6.1 Introduction

Regression tasks are usually cast as modelling the distribution of a vector-valued output $\boldsymbol{y}$ given a vector-valued input $\boldsymbol{x}$ via a deterministic function, such as a neural network, taking $\boldsymbol{x}$ as an input. In this setting, the model is trained on a dataset of input-output pairs, and predictions of the outputs are independent of each other given the inputs. An alternative approach to regression involves using the training data to compute a *distribution over functions* that map inputs to outputs, and using draws from that distribution to make predictions on test inputs. This approach allows for reasoning about multiple functions consistent with the data, and can capture the co-variability in outputs given inputs. In the Bayesian machine learning literature, non-parametric models such as Gaussian Processes (GPs) are popular choices of this approach.

Neural Processes (NPs) [Garnelo et al., 2018a,b] offer an efficient method to modelling a distribution over regression functions, with prediction complexity linear in the context set size. Once trained, they can predict the distribution of an arbitrary *target* output conditioned on a set of *context* input-output pairs of an arbitrary size. This flexibility of NPs enables them to model data that can be interpreted as being generated from a stochastic process. It is important to note however that NPs and GPs have different training regimes. NPs are trained on samples from multiple realisations of a stochastic process (i.e. trained on many different functions), whereas GPs are usually trained on observations from one realisation of the stochastic process (a single function). Hence a direct comparison between the two is usually not plausible.

Despite their many appealing properties, one substantial weakness of NPs is that they tend to underfit the context set. This manifests in the 1D curve fitting example on the left half of Figure 6.1 as inaccurate predictive means and overestimated variances at the input locations of the context set. The right half of the figure shows this phenomenon when predicting the bottom half of a face image from its top half: although the prediction is globally coherent, the model's reconstruction of the top-half is far from perfect. In an NP, the encoder aggregates the context set to a fixed-length latent summary via a permutation invariant function, and the decoder maps the latent and target input to the target output. We hypothesise that the underfitting behaviour is because the mean-aggregation step in the encoder acts as a bottleneck: since taking the mean across context representations gives the same weight to each context point, it is difficult for the decoder to learn which context points provide relevant

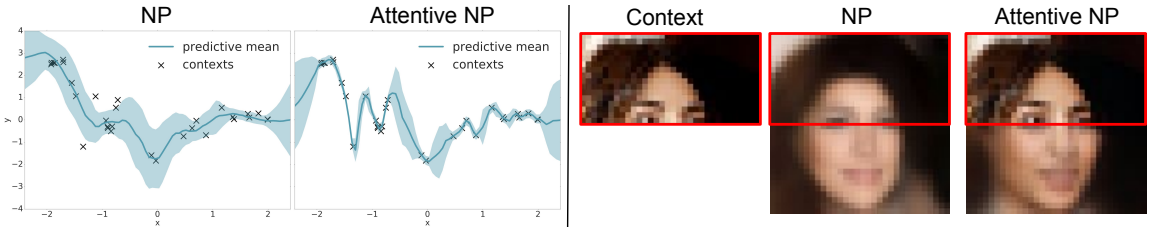


Figure 6.1: Comparison of predictions given by a fully trained NP and Attentive NP (ANP) in 1D function regression (left) / 2D image regression (right). The contexts (crosses/top half pixels) are used to predict the target outputs (y-values of all $x \in [-2, 2]$/all pixels in image). The ANP predictions are noticeably more accurate than for NP at the context points.

information for a given target prediction. In theory, increasing the dimensionality of the representation could address this issue, but we show in Section 6.4 that in practice, this is not sufficient.

To address this issue, we draw inspiration from GPs, which also define a family of conditional distributions for regression. In GPs, the kernel can be interpreted as a measure of similarity among two points in the input domain, and shows which context points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ are relevant for a given query $\boldsymbol{x}_*$. Hence when $\boldsymbol{x}_*$ is close to some $\boldsymbol{x}_i$, its y-value prediction $\boldsymbol{y}_*$ is necessarily close to $\boldsymbol{y}_i$ (assuming small likelihood noise), and there is no risk of underfitting. We implement a similar mechanism in NPs using differentiable attention that learns to attend to the contexts relevant to the given target, while preserving the permutation invariance in the contexts. We evaluate the resulting *Attentive Neural Processes* (ANPs) on 1D function regression and on 2D image regression. Our results show that ANPs greatly improve upon NPs in terms of reconstruction of contexts as well as speed of training, both against iterations and wall clock time. We also demonstrate that ANPs show enhanced expressiveness relative to the NP and is able to model a wider range of functions.

6.2 Background

6.2.1 Neural Processes

The NP is a model for regression functions that map an input $\boldsymbol{x}_i \in \mathbb{R}^{d_x}$ to an output $\boldsymbol{y}_i \in \mathbb{R}^{d_y}$. In particular, the NP defines a (infinite) family of conditional distributions, where one may condition on an arbitrary number of observed *contexts* $(\boldsymbol{x}_C, \boldsymbol{y}_C) \coloneqq (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i \in C}$ to model an arbitrary number of *targets* $(\boldsymbol{x}_T, \boldsymbol{y}_T) \coloneqq (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i \in T}$

in a way that is invariant to ordering of the contexts and ordering of the targets. The model is defined for arbitrary C and T but in practice we use $C \subset T$. The deterministic NP models these conditional distributions as:

$$p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{x}_C, \boldsymbol{y}_C) \coloneqq p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{r}_C) \tag{6.1}$$

with $\boldsymbol{r}_C \coloneqq r(\boldsymbol{x}_C, \boldsymbol{y}_C) \in \mathbb{R}^d$ where r is a deterministic function that aggregates $(\boldsymbol{x}_C, \boldsymbol{y}_C)$ into a finite dimensional representation with permutation invariance in C. In practice, each context $(\boldsymbol{x}, \boldsymbol{y})$ pair is passed through an MLP to form a representation of each pair, and these are aggregated by taking the mean to form $\boldsymbol{r}_C$. The likelihood $p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{r}_C)$ is modelled by a Gaussian factorised across the targets $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i \in T}$ with mean and variance given by passing $\boldsymbol{x}_i$ and $\boldsymbol{r}_C$ through an MLP. The unconditional distribution $p(\boldsymbol{y}_T|\boldsymbol{x}_T)$ (when $C = \emptyset$) is defined by letting $r_\emptyset$ be a fixed vector.

The latent variable version of the NP model includes a global latent $\boldsymbol{z}$ to account for uncertainty in the predictions of $\boldsymbol{y}_T$ for a given observed $(\boldsymbol{x}_C, \boldsymbol{y}_C)$. It is incorporated into the model via a *latent path* that complements the *deterministic path* described above. Here $\boldsymbol{z}$ is modelled by a factorised Gaussian parameterised by $\boldsymbol{s}_C \coloneqq s(\boldsymbol{x}_C, \boldsymbol{y}_C)$, with s being a function of the same properties as r

$$p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{x}_C, \boldsymbol{y}_C) \coloneqq \int p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{r}_C, \boldsymbol{z}) q(\boldsymbol{z}|\boldsymbol{s}_C) d\boldsymbol{z} \tag{6.2}$$

with $q(\boldsymbol{z}|\boldsymbol{s}_\emptyset) \coloneqq p(\boldsymbol{z})$, the prior on $\boldsymbol{z}$. The likelihood is referred to as the *decoder*, and q, r, s form the *encoder*. See Figure 6.2 for diagrams of these models.

The motivation for having a global latent is to model different realisations of the data generating stochastic process — each sample of $\boldsymbol{z}$ would correspond to one realisation of the stochastic process. One can define the model using either just the deterministic path, just the latent path, or both. In this work we investigate the case of using both paths, which gives the most expressive model and also gives a sensible setup for incorporating attention, as we will show later in Section 6.3.

The parameters of the encoder and decoder are learned by maximising the following ELBO

$$\log p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{x}_C, \boldsymbol{y}_C) \geq \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{s}_T)}[\log p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{r}_C, \boldsymbol{z})] - D_{\mathrm{KL}}(q(\boldsymbol{z}|\boldsymbol{s}_T) \| q(\boldsymbol{z}|\boldsymbol{s}_C)) \tag{6.3}$$

for a random subset of contexts C and targets T via the reparameterisation trick [Kingma and Welling, 2013, Rezende et al., 2014]. In other words, the NP learns to reconstruct targets, regularised by a KL term that encourages the summary of the

contexts to be not too far from the summary of the targets. This is sensible since we are assuming that the contexts and targets come from the same realisation of the data-generating stochastic process, and especially so if targets contain contexts. At each training iteration, the number of contexts and targets are also chosen randomly (as well as being randomly sampled from the training data), so that the NP can learn a wide family of conditional distributions.

NPs have many desirable properties, namely (i) **Scalability**: computation scales linearly at $O(n+m)$ for n contexts and m targets at train and prediction time. (ii) **Flexibility**: defines a very wide family of distributions, where one can condition on an arbitrary number of contexts to predict an arbitrary number of targets. (iii) **Permutation invariance**: the predictions of the targets are order invariant in the contexts. However these advantages come at the cost of not satisfying consistency in the contexts. For example, if $\boldsymbol{y}_{1:m}$ is generated given some context set, then its distribution need not match the distribution you would obtain if $\boldsymbol{y}_{1:n}$ is generated first, appended to the context set then $\boldsymbol{y}_{n+1:m}$ is generated. However maximum-likelihood learning can be interpreted as minimising the KL between the (consistent) conditional distributions of the data-generating stochastic process and the corresponding conditional distributions of the NP. Hence we could view the NP as approximating the conditionals of the consistent data-generating stochastic process.

6.2.2 Attention

Given a set of key-value pairs $(k_i, v_i)_{i\in\mathcal{I}}$ and a query q, an attention mechanism computes weights of each key with respect to the query, and aggregates the values with these weights to form the value corresponding to the query. In other words, the query *attends* to the key-value pairs. The queried values are invariant to the ordering of the key-value pairs; this permutation invariance property of attention is key in its application to NPs. The idea of using a differentiable addressing mechanism that can be learned from the data has been applied successfully in various areas of Deep Learning, namely handwriting generation and recognition [Graves, 2012] and neural machine translation [Bahdanau et al., 2014]. More recently, there has been work employing *self-attention* (where keys and queries are identical) to give expressive sequence-to-sequence mappings in natural language processing [Vaswani et al., 2017] and image modelling [Parmar et al., 2018].

We give some examples of attention mechanisms which are used in the paper. Suppose we have n key-value pairs arranged as matrices $K \in \mathbb{R}^{n \times d_k}$, $V \in \mathbb{R}^{n \times d_v}$, and m queries $Q \in \mathbb{R}^{m \times d_k}$. Simple forms of attention based on locality (weighting keys according to distance from query) are given by various stationary kernels. For example, the (normalised) *Laplace* kernel gives the queried values as

$$\textbf{Laplace}(Q, K, V) \coloneqq WV \in \mathbb{R}^{m \times d_v}, \quad W_{i\cdot} \coloneqq \text{softmax}((-||Q_{i\cdot} - K_{j\cdot}||_1)_{j=1}^n) \in \mathbb{R}^n$$

Similarly (scaled) *dot-product* attention uses the dot-product between the query and keys as a measure of similarity, and weights the keys according to the values

$$\textbf{DotProduct}(Q, K, V) \coloneqq \text{softmax}(QK^\top / \sqrt{d_k})V \in \mathbb{R}^{m \times d_v}$$

The use of dot-product attention allows the query values to be computed with two matrix multiplications and a softmax, allowing for use of highly optimised matrix multiplication code.

Multihead attention [Vaswani et al., 2017] is a parameterised extension where for each head, the keys, values and queries are linearly transformed, then dot-product attention is applied to give head-specific values. These values are concatenated and linearly transformed to produce the final values:

$$\begin{aligned} \textbf{MultiHead}(Q, K, V) &\coloneqq \text{concat}(\text{head}_1, \ldots, \text{head}_H)W \in \mathbb{R}^{m \times d_v} \\ \text{where head}_h &\coloneqq \text{DotProduct}(QW_h^Q, KW_h^K, VW_h^V) \in \mathbb{R}^{m \times d_v} \end{aligned}$$

This multihead architecture allows the query to attend to different keys for each head and tends to give smoother query-values than dot-product attention (c.f. Section 6.4).

6.3 Attentive Neural Processes

Figure 6.2 describes how attention is incorporated into NP to give the Attentive NP (ANP). In summary, *self-attention* is applied to the context points to compute representations of each $(\boldsymbol{x}, \boldsymbol{y})$ pair, and the target input attends to these context representations (*cross-attention*) to predict the target output. In detail, the representation of each context pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i \in C}$ before the mean-aggregation step is computed by a self-attention mechanism, in both the deterministic and latent path. The intuition for the self-attention is to model interactions between the context points. For example, if many context points overlap, then the query need not attend to all of

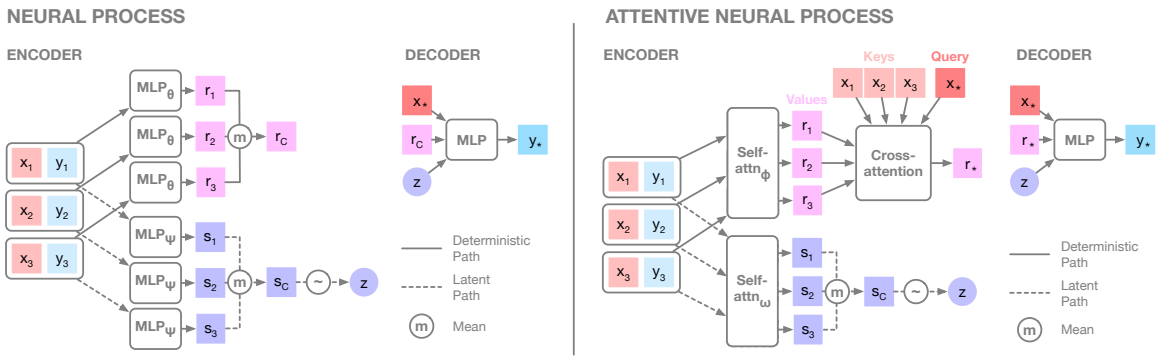


Figure 6.2: Model architecture for the NP (left) and Attentive NP (right)

these points, but only give high weight to one or a few. The self-attention will help obtain richer representations of the context points that encode these types of relations between the context points. We model higher order interactions by simply stacking the self-attention, as is done in Vaswani et al. [2017].

In the deterministic path, the mean-aggregation of the context representations that produces $\boldsymbol{r}_C$ is replaced by a *cross-attention* mechanism, where each target query $\boldsymbol{x}_*$ attends to the context $\boldsymbol{x}_C \coloneqq (\boldsymbol{x}_i)_{i \in C}$ to produce a query-specific representation $\boldsymbol{r}_* \coloneqq r^*(\boldsymbol{x}_C, \boldsymbol{y}_C, \boldsymbol{x}_*)$. This is precisely where the model allows each query to attend more closely to the context points that it deems relevant for the prediction. The reason we do not have an analogous mechanism in the latent path is that we would like to preserve the global latent, that induces dependencies between the target predictions. The interpretation of the latent path is that $\boldsymbol{z}$ gives rise to correlations in the marginal distribution of the target predictions $\boldsymbol{y}_T$, modelling the global structure of the stochastic process realisation, whereas the deterministic path models the fine-grained local structure.

The decoder remains the same, except we replace the shared context representation $\boldsymbol{r}_C$ with the query-specific representation $\boldsymbol{r}_*$. Note that permutation invariance in the contexts is preserved with the attention mechanism. If we use uniform attention (all contexts given the same weight) throughout, we recover the NP. ANP is trained using the same loss (6.3) as the original NP, also using Gaussian likelihood $p(\boldsymbol{y}_i | \boldsymbol{x}_i, r^*(\boldsymbol{x}_C, \boldsymbol{y}_C, \boldsymbol{x}_i), \boldsymbol{z})$ and diagonal Gaussian $q(\boldsymbol{z} | \boldsymbol{s}_C)$.

The added expressivity and resulting accuracy of the NP with attention comes at a cost. The computational complexity is raised from $O(n + m)$ to $O(n(n + m))$, since we apply self-attention across the contexts and for every target point we compute weights for all contexts. However most of the computation for the (self-)attention

is done via matrix multiplication (c.f. Section 6.2.2), and so can be done in parallel across the contexts and across the targets. In practice, the training time for ANPs remains comparable to NPs, and in fact we show that ANPs learn significantly faster than NPs not only in terms of training iterations but also in wall-clock time, despite being slower at prediction time (c.f. Section 6.4).

6.4 Experimental Results

Note that the (A)NP learns a stochastic process, so should be trained on multiple functions that are realisations of the stochastic process. At each training iteration, we draw a batch of realisations from the data generating stochastic process, and select random points on these realisations to be the targets and a subset to be the contexts to optimise the loss in Equation (6.3). We use the same decoder architecture for all experiments, and 8 heads for multihead. See Appendix 6.7.1 for architectural details.

1D Function regression on synthetic GP data We first explore the (A)NPs trained on data that is generated from a Gaussian Process with a squared-exponential kernel and small likelihood noise[1]. We emphasise that (A)NPs need not be trained on GP data or data generated from a known stochastic process, and this is just an illustrative example. We explore two settings: one where the hyperparameters of the kernel are fixed throughout training, and another where they vary randomly at each training iteration. The number of contexts (n) and number of targets (m) are chosen randomly at each iteration ($n \sim U[3, 100]$, $m \sim n + U[0, 100 - n]$). Each x-value is drawn uniformly at random in $[-2, 2]$. For this simple 1D data, we do not use self-attention and just explore the use of cross-attention in the deterministic path (c.f. Figure 6.2). Thus we use the same encoder/decoder architecture for NP and ANP, except for the cross-attention. See Appendix 6.7.2 for experimental details.

Figure 6.3 (left) shows context reconstruction error $\frac{1}{|C|}\sum_{i\in C}\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{s}_C)}[\log p(\boldsymbol{y}_i|\boldsymbol{x}_i, r^*(\boldsymbol{x}_C, \boldsymbol{y}_C, \boldsymbol{x}_i), \boldsymbol{z})]$ and NLL of targets given contexts $\frac{1}{|T|}\sum_{i\in T}\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{s}_C)}[\log p(\boldsymbol{y}_i|\boldsymbol{x}_i, r^*(\boldsymbol{x}_C, \boldsymbol{y}_C, \boldsymbol{x}_i), \boldsymbol{z})]$ for the different attention mechanisms, trained on a GP with random kernel hyperparameters. ANP shows a much more rapid decrease in reconstruction error and lower values at convergence compared to the NP, especially for dot product and multihead attention. This holds not only against training iteration but also against wall clock time, so learning is fast despite

[1]Code is available at `https://github.com/deepmind/neural-processes/blob/master/attentive_neural_process.ipynb`

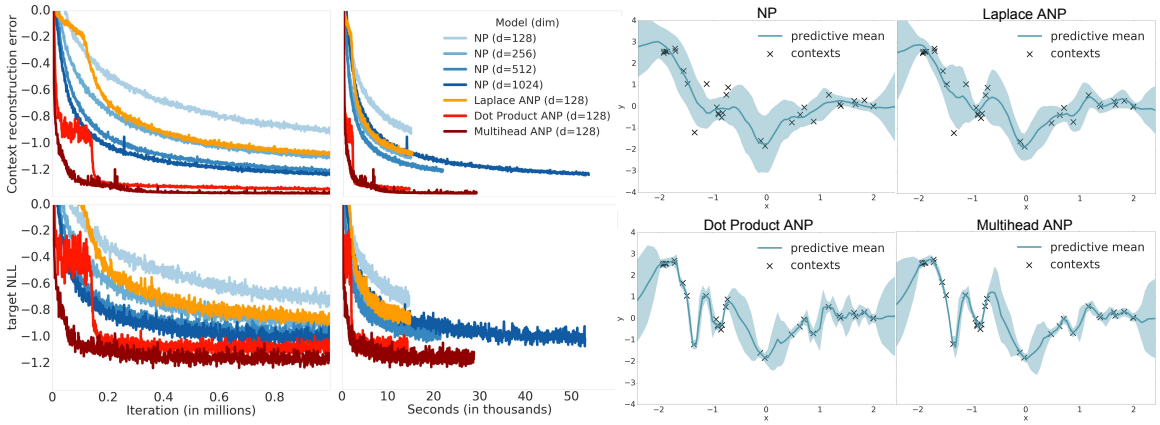


Figure 6.3: Qualitative and quantitative results of different attention mechanisms for 1D GP function regression with random kernel hyperparameters. **Left**: moving average of context reconstruction error (top) and target negative log likelihood (NLL) given contexts (bottom) plotted against training iterations (left) and wall clock time (right). d denotes the bottleneck size i.e. hidden layer size of all MLPs and the dimensionality of r and z. **Right**: predictive mean and variance of different attention mechanisms given the same context. Best viewed in colour.

the added computational cost of attention. The right column plots show that the computation times of Laplace and dot-product ANP are similar to the NP for the same value of d, and multihead ANP takes around twice the time. We also show how the size of the bottleneck (d) in the deterministic and latent paths of the NP affects the underfitting behaviour of NPs. The figure shows that raising d does help achieve better reconstructions, but there appears to be a limit in how much reconstructions can improve. Beyond a certain value of d, the learning for the NP becomes too slow, and the value of reconstruction error at convergence is still higher than that achieved by multihead ANP with 10% of the wall-clock time. Hence using ANPs has significant benefits over simply raising the bottleneck size in NPs.

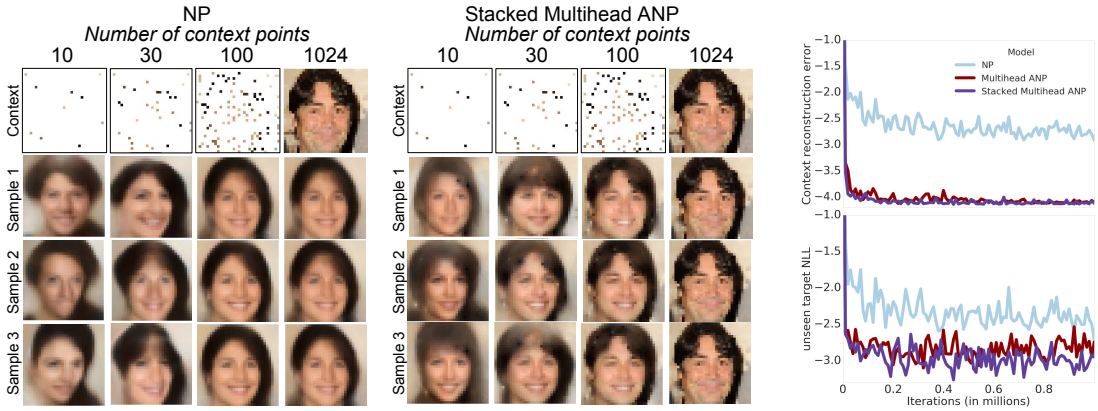
In Figure 6.3 (right) we visualise the learned conditional distribution for a qualitative comparison of the attention mechanisms. The context is drawn from the GP with the hyperparameter values that give the most fluctuation. Note that the predictive mean of the NP underfits the context, and tries to explain the data by learning a large likelihood noise. Laplace shows similar behaviour, whereas dot-product attention gives predictive means that accurately predict almost all context points. Note that Laplace attention is parameter-free (keys and queries are the x-coordinates) whereas for dot-product attention we have set the keys and queries to be parameterised representations of the x-values (output of learned MLP that takes x-coordinates as inputs). So the dot-product similarities are computed in a learned representation space,

whereas for Laplace attention the similarities are computed based on L1 distance in the x-coordinate domain, hence it is expected that dot-product attention outperforms Laplace attention. However dot-product attention displays non-smooth predictions, shown more clearly in the predictive standard deviations (c.f. Appendix 6.7.3 for an explanation). The multiple heads in multihead attention appear to help smooth out the interpolations, giving good reconstruction of the contexts as well as prediction of the targets, while preserving increased predictive uncertainty away from the contexts as in a GP. The results for (A)NP trained on fixed GP kernel hyperparameters are similar (c.f. Appendix 6.7.3), except that the NP underfits to a lesser degree because of the reduced variety of sample curves (functions) in the data. This difference in performance for the two kernel hyperparameter settings provides evidence of how the ANP is more expressive than the NP and can learn a wider range of functions.

Using the trained (A)NPs we tackle a toy Bayesian Optimisation (BO) problem, where the task is to find the minimum of test functions drawn from a GP prior. This is a proof-of-concept experiment showing the utility of being able to sample entire functions from the (A)NP and having accurate context reconstructions. See Appendix 6.7.3 for the details and an analysis of results.

2D Function regression on image data Image data can also be interpreted as being generated from a stochastic process (since there are dependencies between pixel values), and predicting the pixel values can be cast as a regression problem mapping a 2D pixel location $\boldsymbol{x}_i$ to its pixel intensity $\boldsymbol{y}_i$ ($\in \mathbb{R}^1$ for greyscale, $\in \mathbb{R}^3$ for RGB). Each image corresponds to one realisation of the process sampled on a fixed 2 dimensional grid. We train the ANP on MNIST [LeCun et al., 1998] and 32×32 CelebA [Liu et al., 2015] using the standard train/test split with up to 200 context/target points at training. For this application we explore the use of self-attentional layers in the encoder, stacking them as is done in Parmar et al. [2018]. See Appendix 6.7.4 for experimental details.

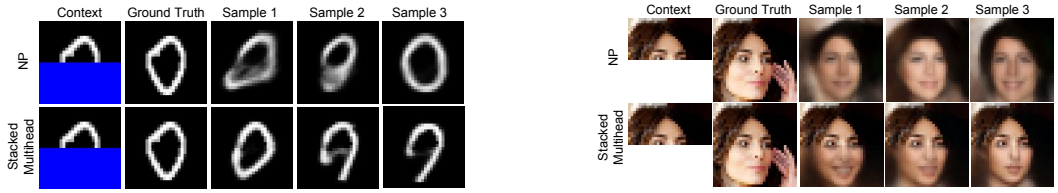
On both datasets we show results of three different models: NP, ANP with multihead cross-attention in the deterministic path (*Multihead* ANP), and ANP with both multihead attention in the deterministic path and two layers of stacked self-attention in both the deterministic and latent paths (*Stacked Multihead* ANP). Figure 6.4a shows predictions of the full image (i.e. full target) with a varying number of random context pixels, from 10 to 1024 (full image) for a randomly selected image (see Appendix 6.7.5 for other images). For each we generate predictions that correspond to the mean of $p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{r}_C, \boldsymbol{z})$ for three different samples of $\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{s}_C)$. The NP (left) gives



(a) Reconstructions of full CelebA image from a varying number of random context points for NP (left) and *Stacked Multihead* ANP (right).

(b) Context NLL (top) and unseen target NLL given contexts (bottom).

Figure 6.4: Qualitative and quantitative results on test set for 2D CelebA function regression.



(a) MNIST

(b) CelebA

Figure 6.5: Reconstruction of full image from top half. The CelebA results use the **same models** (with the **same parameter values**) as Figure 6.4a.

reasonable predictions with a fair amount of diversity for fewer contexts, but the reconstructions of the whole image are not accurate, compared to *Stacked Multihead* ANP (right) where the reconstructions are indistinguishable from the original. The use of attention also helps achieve crisper inpaintings when the target pixels are filled in, enhancing the ANP's ability to model less smooth 2D functions compared to the NP. The diversity in faces and digits obtained with different values of $\boldsymbol{z}$ is apparent the different samples, providing evidence for the claim that $\boldsymbol{z}$ can model global structure of the image, with one sample corresponding to one realisation of the data generating stochastic process. Similar conclusions hold for MNIST (see Appendix 6.7.5) and for the full image prediction using the top half as context in Figure 6.5. In the latter task, note that the model has never been trained on more than 200 context points, yet it manages to generalise to when the context is of size 512 (half the image).

Figure 6.4b verifies quantitatively that both *Multihead* and *Stacked Multihead* ANP

Figure 6.6: Pixels attended to by each head of multihead attention in *Multihead* ANP given a target pixel. Each head is given a different colour and the target pixel is marked with a cross.

give a much improved context reconstruction error compared to the NP. Similarly the NLL for the target points (that are not included in the context) is improved with multihead cross-attention, showing small gains with stacked self-attention. However qualitatively, there are noticeable gains in crispness and global coherence when using stacked self-attention (see Appendix 6.7.5).

In Figure 6.6 we visualise each head of *Multihead* ANP for CelebA. We let the target pixel (cross) attend to all pixels, and see where each head of the attention focuses on. We colour-code the pixels with the top 20 weights per head, with intensity proportional to the attention weight. We can see that each head has different roles: the cyan head only looks at the target pixel and nothing else; the red head looks at a few pixels nearby; the green head looks at a larger region nearby; the yellow looks at the pixels on the column of the target; the orange looks at some band of the image; the purple head (interestingly) looks at the other side of the image, trying to exploit the symmetry of faces. We observe consistent behaviour in these heads for other target pixels (see Figure 6.16 of Appendix 6.7.5).

One other illustrative application of (A)NPs trained on images is that one can map images from one resolution to another, even if the model has only been trained on one resolution. Because the two dimensional $\boldsymbol{x}$ (pixel locations) are modelled as real values that live in a continuous space, the model can predict the $\boldsymbol{y}$ (pixel intensities)

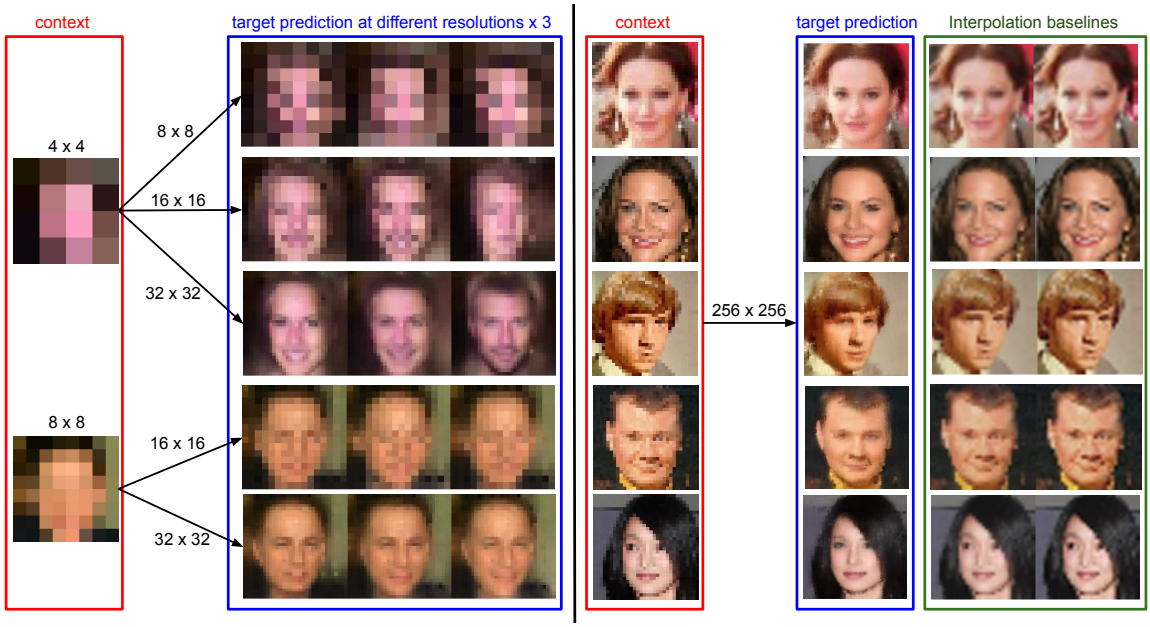


Figure 6.7: Mapping between different resolutions by the **same model** (with the **same parameter values**) as *Stacked Multihead* ANP in Figures 6.4a, 6.5b. The two rightmost columns show the results of baseline methods, namely linear and cubic interpolation to 256×256.

of any point in this space, and not just the grid of points that it was trained on. Hence using one grid as the context and a finer grid as the target, the model can map a given resolution to a higher resolution. This could, however, be problematic for NPs whose reconstructions can be inaccurate, so the prediction of the target resolution can look very different to the original image (see Figure 6.19 of Appendix 6.7.5). The reconstructions of ANPs may be accurate enough to give reliable mappings between different resolutions. We show results for such mappings given by the same *Stacked Multihead* ANP (the same model used to produce Figures 6.4a, 6.5b) in Figure 6.7. On the left, we see that the ANP (trained on 32×32 images) is capable of mapping low resolutions (4×4 or 8×8) to fairly realistic 32×32 target outputs with some diversity for different values of $\boldsymbol{z}$ (more diversity for the 4×4 contexts as expected). Perhaps this performance is to be expected since the model has been trained on data that has 32×32 resolution. The same model allows us to map to even higher resolutions, namely from the original 32×32 images to 256×256, displayed on the right of the figure. We see that even though the model has never seen any images beyond the original resolution, the model learns a fairly realistic high resolution image with sharper edges compared to the baseline interpolation methods. Moreover, there is some evidence that it learns an internal representation of the appearance of faces,

when for example it learns to fill in the eye even when the original image is too coarse to separate the iris (coloured part) from the sclera (white part) (e.g. top row image), a feature that is not possible with simple interpolation. See Figure 6.19 in Appendix 6.7.5 for larger versions of the images.

For each of MNIST and CelebA, all qualitative plots in this section were given from the **same model** (with the **same parameter values**) for each attention mechanism, learned by optimising the loss in Equation (6.3) over random context pixels and random target pixels at each iteration. It is important to note that we do not claim the ANP to be a replacement of state of the art algorithms of image inpainting or super-resolution, and rather we show these image applications to highlight the flexibility of the ANP in modelling a wide family of conditional distributions.

6.5 Related Work

The work related to NPs in the domain of Gaussian Processes, Meta-Learning, conditional latent variable models and Bayesian Learning have been discussed extensively in the original works of Garnelo et al. [2018a,b], hence we focus on works that are particularly relevant for ANPs.

Gaussian Processes (GPs) Returning to our motivation for using attention in NPs, there is a clear parallel between GP kernels and attention, in that they both give a measure of similarity between two points in the same domain. The use of attention in an embedding space that we explore is related to Deep Kernel Learning [Wilson et al., 2016] where a GP is applied to learned representations of data. Here, however, learning is still done in a GP framework by maximising the marginal likelihood. We reiterate that the training regimes of GPs and NPs are different, so a direct comparison between the methods is difficult. One possibility for comparison is to learn the GP via the training regime of NPs, namely updating the kernel hyperparameters at each iteration via one gradient step of the marginal likelihood on the mini-batch of data. However, this would still have a $O(n^3)$ computational cost in the naive setting and may require kernel approximations. In general, the predictive uncertainties of GPs depend heavily on the choice of the kernel, whereas NPs learn predictive uncertainties directly from the data. Despite these drawbacks, GPs have the benefit of being consistent stochastic processes, and the covariance between the predictions at different x-values and the marginal variance of each prediction can be expressed exactly in closed form, a feature that the current formulation of (A)NPs do not have. Variational Implicit Processes

(VIP) [Ma et al., 2018] are also related to NPs, where VIP defines a stochastic process using the same decoder setup with a finite dimensional $\boldsymbol{z}$. Here, however, the process and its posterior given observed data are both approximated by a GP and learned via a generalisation of the Wake-Sleep algorithm [Hinton et al., 1995].

Meta-Learning (A)NPs can be seen as models that do few-shot learning, although this is not the focus of our work. Given input-output pairs drawn from a new function at test time, one can reason about this function by looking at the predictive distribution conditioning on these input-output pairs. There is a plethora of works in few-shot classification, of which Vinyals et al. [2016], Snell et al. [2017], Santoro et al. [2016] use attention to locate the relevant observed image/prototype given a query image. Attention has also been used for tasks in Meta-RL such as continuous control and visual navigation [Mishra et al., 2018]. Few-shot density estimation using attention has also been explored extensively in numerous works [Rezende et al., 2016, Reed et al., 2017, Bornschein et al., 2017, Bartunov and Vetrov, 2018]. Especially relevant are the Neural Statistician [Edwards and Storkey, 2017] and the Variational Homoencoder [Hewitt et al., 2018] who have a similar permutation invariant encoder (that outputs summaries of a data set), but use local latents on top of a global latent. For ANPs, we look at the less-explored regression setting. The authors of Vfunc [Bachman et al., 2018] also explore regression on a toy 1D domain, using a similar setup to NPs but optimising an approximation to the entropy of the latent function, without any attention mechanisms. Multi-task learning has also been tackled in the GP literature by various works [Teh et al., 2005, Bonilla et al., 2008, Alvarez et al., 2012, Dai et al., 2017].

Generative Query Networks [Eslami et al., 2018, Kumar et al., 2018b] are models for spatial prediction that render a frame of a scene given a viewpoint. Their model corresponds to a special case of NPs where the $\boldsymbol{x}$ are viewpoints and the $\boldsymbol{y}$ are frames of a scene. Rosenbaum et al. [2018] apply the GQN to the task of 3D localisation with an attention mechanism, but attention is applied to patches of context frames ($\boldsymbol{y}$) instead of a parametric representation of viewpoints ($\boldsymbol{x}$). Note that in our work the targets attend to the contexts via the $\boldsymbol{x}$.

6.6 Conclusion and Discussion

We have proposed ANPs, which augment NPs with attention to resolve the fundamental problem of underfitting. We have shown that this greatly improves the accuracy of

predictions in terms of context and target NLL, results in faster training, and expands the range of functions that can be modelled. There is a wide scope of future work for ANPs. Regarding model architecture, one way of incorporating cross-attention into the latent path and modelling the dependencies across the resulting local latents is to also have a global latent, much like the setup of the Neural Statistician but translated to the regression setting. An interesting further application would be to train ANPs on text data, enabling them to fill in the blanks in a stochastic manner. For the image application, the Image Transformer (ImT) [Parmar et al., 2018] has some interesting connections with ANPs: its local self-attention used to predict consecutive pixel blocks from previous blocks has parallels with how our model attends to context pixels to predict target pixels. Replacing the MLP in the decoder of the ANP with self-attention across the target pixels, we have a model that closely resembles an ImT defined on arbitrary orderings of pixels. This is in contrast to the original ImT, which presumes a fixed ordering and is trained autoregressively. We plan to equip ANPs with self-attention in the decoder, and see how far their expressiveness can be extended. In this setup, however, the targets will affect each other's predictions, so the ordering and grouping of the targets will become important.

Acknowledgements

We would like to thank Ali Razavi for his advice on implementing multihead attention, and Michael Figurnov for helpful discussion.

6.7 Supplementary Material

6.7.1 Architectural details for (A)NP

We show the architectural details of the NP and the Multihead ANP models used for the 1D and 2D regression experiments below in Figure 6.8. All MLPs have relu non-linearities except the final layer, which has no non-linearity. The latent path outputs $\mu_z, \omega_z \in \mathbb{R}^d$, which parameterises $q(\boldsymbol{z}|\boldsymbol{s}_C) = \mathcal{N}(\boldsymbol{z}|\mu_z, 0.1 + 0.9\sigma(\omega_z))$ where σ is the sigmoid function. Similarly the decoder outputs μ_y, ω_y, which parameterises $p(\boldsymbol{y}_i|\boldsymbol{z}, \boldsymbol{x}_C, \boldsymbol{y}_C, \boldsymbol{x}_i) = \mathcal{N}(\boldsymbol{y}_i|\mu_y, 0.1 + 0.9f(\omega_y))$ where f is the softplus function.

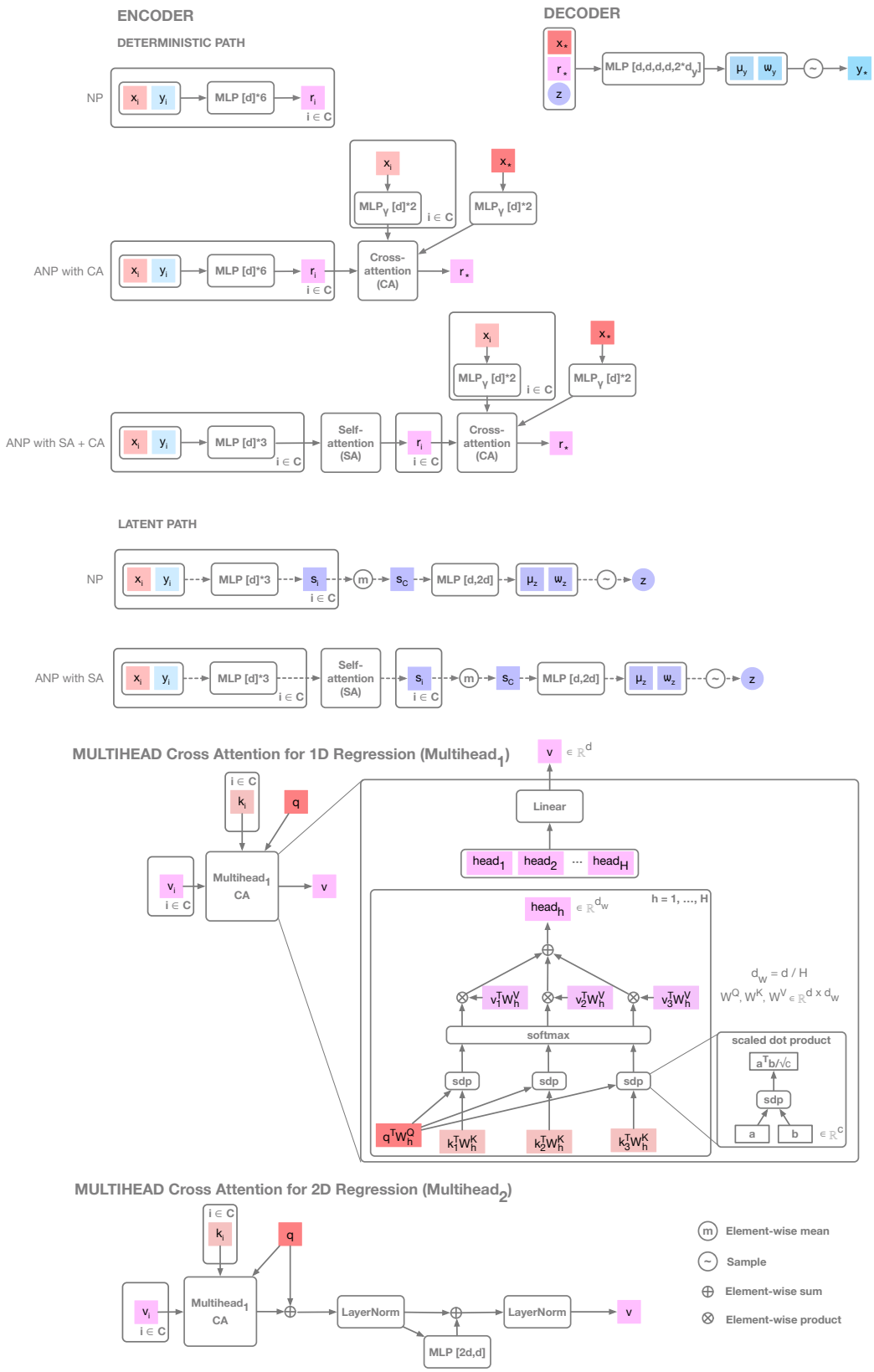


Figure 6.8: The model architecture for NP and ANP for both 1D and 2D regression.

The 1D regression experiments use the basic formulation of multihead cross-attention (denoted $Multihead_1$) in Figure 6.8, whereas the 2D regression experiments uses a form of multihead cross-attention used in the Image Transformer [Parmar et al., 2018]. The only difference is that we do not use dropout, to limit the stochasticity of the model to the latent z.

Self-attention uses the same architecture as cross-attention but with $k_i = v_i$, $q = k_j$ for each $j \in C$, to output $|C|$ representations given $|C|$ input representations. Since the self-attention module has the same number of inputs and outputs, it can be stacked. We stack 2 layers of self-attention for *Stacked Multihead ANP* in the 2D Image regression experiments. Stacking more layers did not lead to noticeable gains qualitatively and quantitatively.

6.7.2 Experimental details of 1D function regression experiment

For the squared exponential kernel of the data generating GP, we use a length scale $l = 0.6$ and kernel scale $\sigma_f^2 = 1$ for the fixed kernel hyperparameter experiments. For the random kernel hyperparameter case, we sample $l \sim U[0.1, 0.6]$, $\sigma_f \sim U[0.1, 1]$. For both, the likelihood noise is $\sigma_n = 0.02$. We use a batch size of 16 — in the fixed hyperparameter setting, we draw 16 curves from a GP with these hyperparameters, and in the random hyperparameter setting, we sample 16 random values of hyperparameters and draw a curve from GPs with each of these hyperparameters. We use the Adam Optimiser [Kingma and Ba, 2015] with a fixed learning rate of 5e-5 and Tensorflow defaults for the other hyperparameters. We use one sample of $q(z|\boldsymbol{s}_C)$ to form a MC estimate of the loss in Equation (6.3) during training and evaluation.

For NP, d is varied between $\{128, 256, 512, 1024\}$ whereas for ANP we always use $d = 128$.

6.7.3 Additional figures for 1D regression on GP data

In Figure 6.9 we also compare the trained (A)NP models against the oracle GP from which the contexts were drawn. We see that the predictions Multihead ANP is notably closer to that of the oracle GP than the NP, but still underestimates the predictive variance. One possible explanation for this is that variational inference (used for

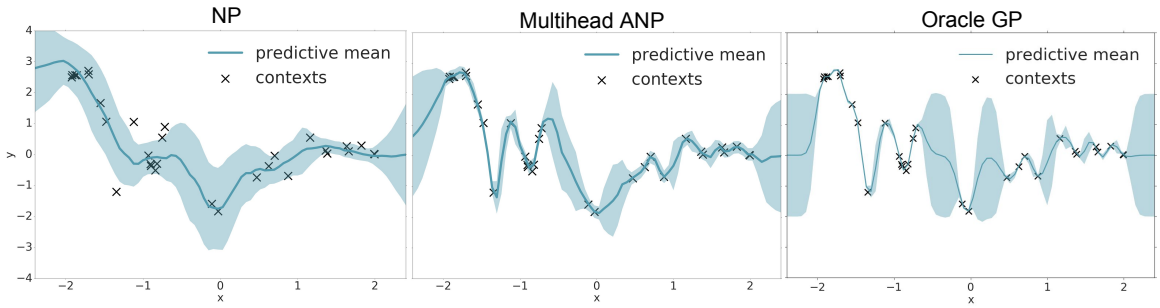


Figure 6.9: Same as right of Figure 6.3 but also comparing against the oracle GP from which context was drawn.

learning the ANP) usually leads to underestimates of predictive variance. It would be interesting to investigate how this issue can be addressed.

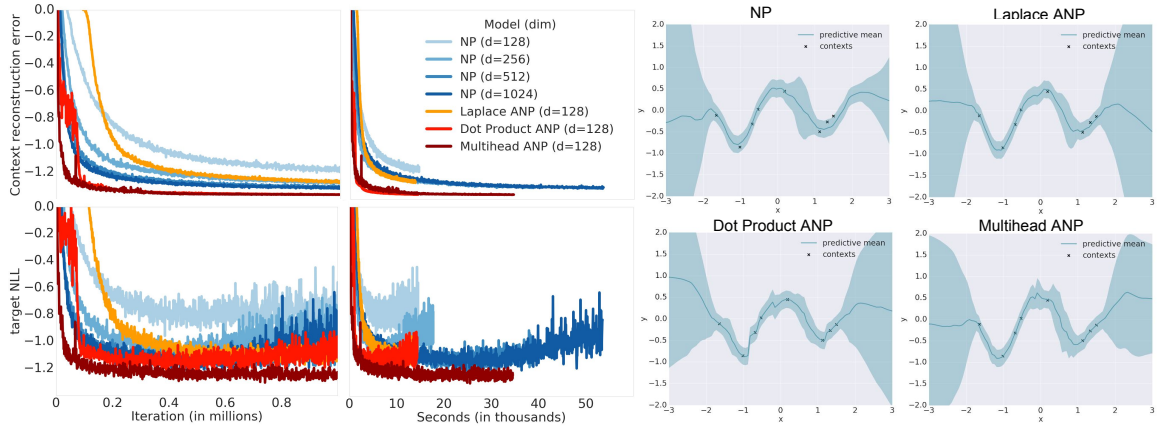


Figure 6.10: Same as Figure 6.3 but for fixed kernel hyperparameters.

The right of Figure 6.10 shows the conditional distributions for fixed kernel hyperparameters (with contexts drawn from the GP with these kernel hyperparameters), with highly non-smooth behaviour for dot-product attention as with the random kernel hyperparameter case. This behaviour seems to arise when the dot-product attention collapses to the local minimum of learning to be a nearest neighbour predictor (with one entry of the softmax becoming saturated), hence giving good reconstructions but poor interpolations between context points.

Figure 6.11 shows how the KL term in the (A)NP loss differs between training on the fixed kernel hyperparameter GP data and on the random kernel hyperparameter GP data. In the fixed hyperparameter case, the KL for multihead ANP quickly goes to 0, indicating that the model deems the deterministic path sufficient to make accurate predictions. However in the random hyperparameter case, there is added variation in

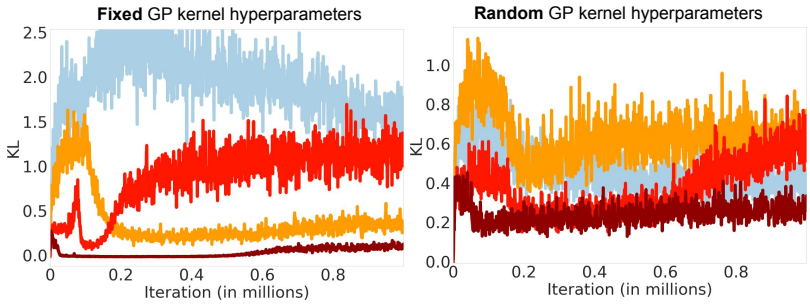


Figure 6.11: KL term in NP loss throughout training for data generated from a GP with fixed (left) and random (right) kernel hyperparameters, using the same colour scheme as Figure 6.10.

the data, hence the attention gives a non-zero KL and uses the latents to model the uncertainty in the realisation of the stochastic process given some context points. In other words, given a context set, the model believes that there are multiple realisations of the stochastic process that can explain these contexts well, hence uses the latents to model this variation.

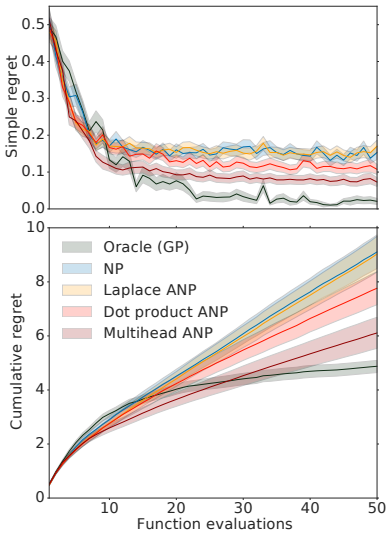


Figure 6.12: Simple and cumulative regret for BO.

Using the same (A)NPs trained on the 1D GP data, we tackle the BO problem of finding the minimum of test functions drawn from a GP prior. We compare ANPs trained with different attention mechanisms to an oracle GP for which we set the kernel hyperparameters to their true value. (A)NPs can be used for BO by considering all previous function evaluations as context points, thus obtaining an informed surrogate of the target function. While other choices are possible, we use Thompson sampling to drawing a simple function from the surrogate and acting according to its minimal predicted value. We show results averaged over 100 test functions in Figure 6.12. We can see that the simple regret (the difference between the predicted and true minimum) is consistently smallest for a NP with multihead attention, approaching the oracle GP. Among the NPs, the slope of the cumulative regret (simple regret summed up to given iteration) decreases most rapidly for multihead, indicating that previous function evaluations are being put to good use for subsequent predictions of the function minimum. The reason that the cumulative regret is initially lower than

the oracle GP is a consequence of under-exploration, due to the uncertainties of ANP away from the context being smaller than that of the oracle GP.

6.7.4 Experimental details of 2D Image regression experiment

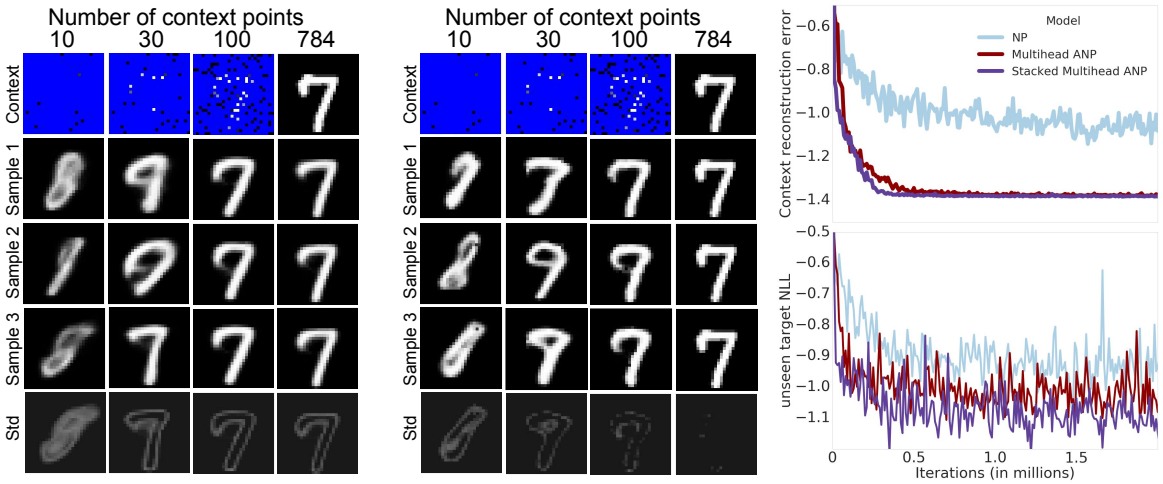
Analogous to the 1D experiments, we take random pixels of a given image at training as targets, and select a subset of this as contexts, again choosing the number of contexts and targets randomly ($n \sim U[3, 200]$, $m \sim n + U[0, 200 - n]$). The $\boldsymbol{x}$ are rescaled to $[-1, 1]$ and the $\boldsymbol{y}$ are rescaled to $[-0.5, 0.5]$. We use a batch size of 16 for both MNIST and CelebA, i.e. use 16 randomly selected images for each batch. We use a learning rate of 5e-5 and 4e-5 respectively for MNIST and CelebA using the Adam optimiser with Tensorflow defaults for the other hyperparameters. The stacked self-attention architecture is the same as in the Image Transformer [Parmar et al., 2018], except that we do not use Dropout to restrict the stochasticity of the model to the global latent $\boldsymbol{z}$, and do not use positional embeddings of the pixels. We use the same architecture for both Mnist and CelebA, and highlight that little tuning has been done regarding the architectural hyperparameters. We again use one sample of $q(z|\boldsymbol{s}_C)$ to form a MC estimate of the loss in Equation (6.3) during training and evaluation.

6.7.5 Additional figures for 2D Image regression on MNIST and CelebA

We can see visually that the NP overestimates the predictive variance by looking at the plot of the standard deviation (bottom row) of Figure 6.13a. We see that the original NP shows noticeable uncertainty around the edges of the reconstruction for all context sets, whereas for the NP with attention, the uncertainty is reduced significantly as you increase the number of contexts until it almost disappears for the full context.

From Figures 6.14 and 6.15 we see that *Stacked Multihead* ANP improves results significantly over *Multihead* ANP, giving sharper images with better global coherence even in the case where the face isn't axis-aligned (see Figure 6.15a).

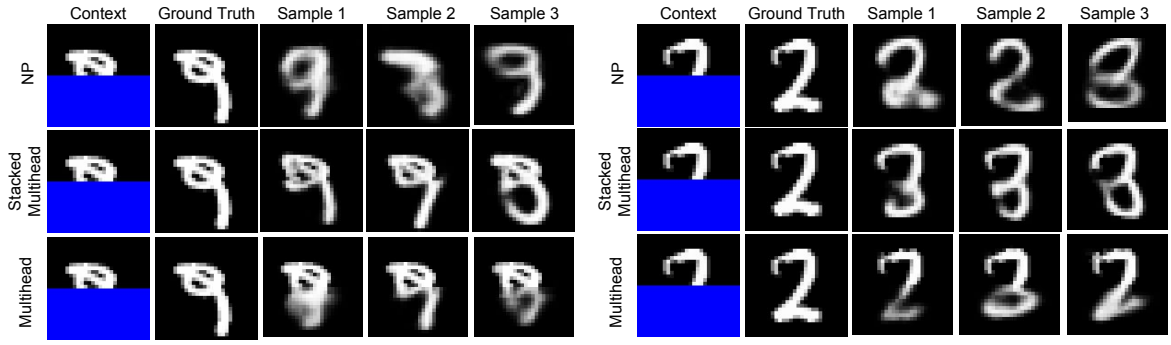
Note that in Figure 6.6, the contexts contain the target, relying on the cyan head would be enough to give an accurate prediction, but the different roles of these heads also hold in the case where the target is disjoint from the context. This is shown



(a) Same as Figure 6.4a but for MNIST.

(b) Same as Figure 6.4b but for MNIST.

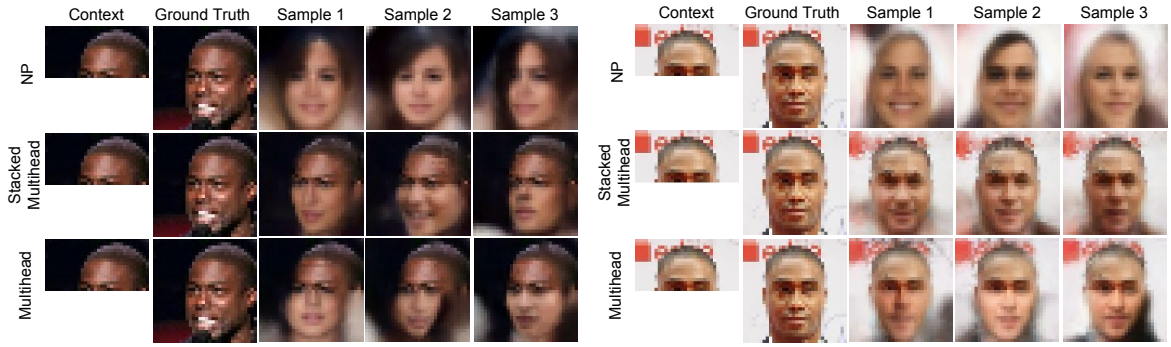
Figure 6.13: Qualitative and quantitative results of different attention mechanisms on test set for 2D MNIST function regression.



(a)

(b)

Figure 6.14: More MNIST reconstruction of full image from top half.



(a)

(b)

Figure 6.15: More CelebA reconstruction of full image from top half.

in Figure 6.16 where the context is disjoint from the target. Here all heads become useful for the target prediction.

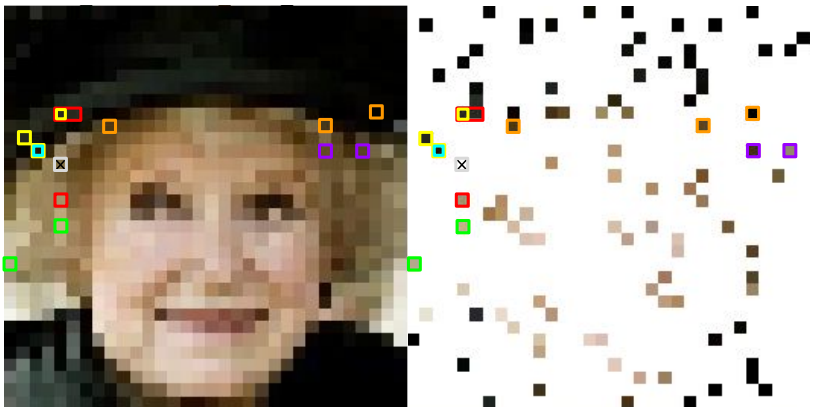


Figure 6.16: Visualisation of pixels attended by each head of multihead attention in the NP given a target pixel and a separate context of 100 random pixels. Each head is given a different colour (consistent with the colours in Figure 6.6 and the target pixel is marked by a cross.

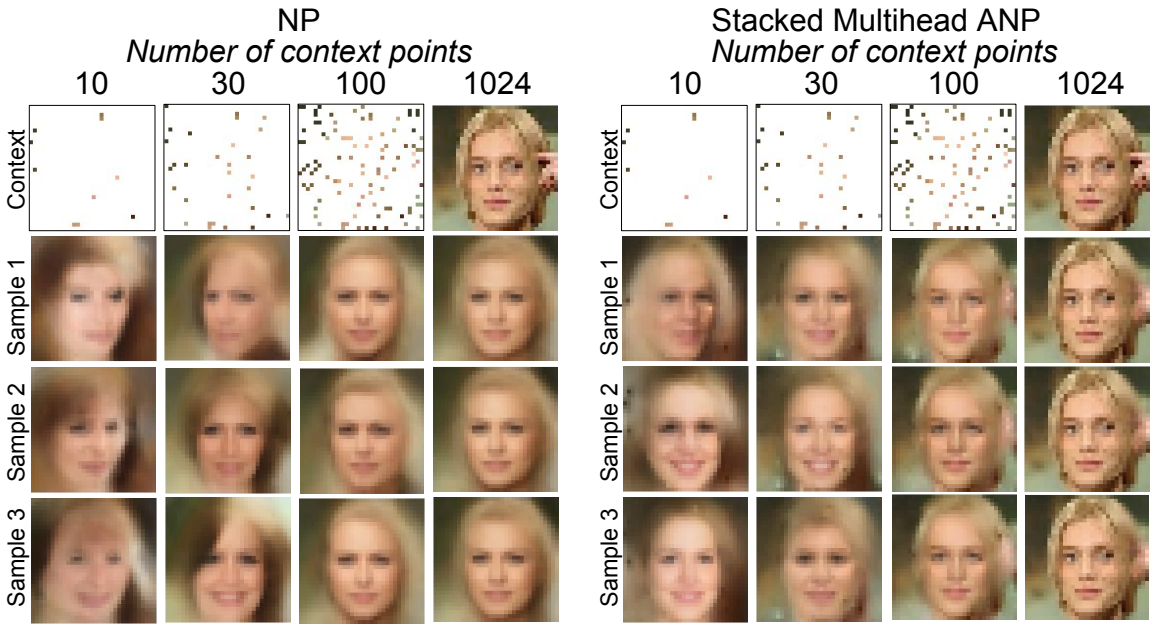


Figure 6.17: Same as Figure 6.4a but for a different image.

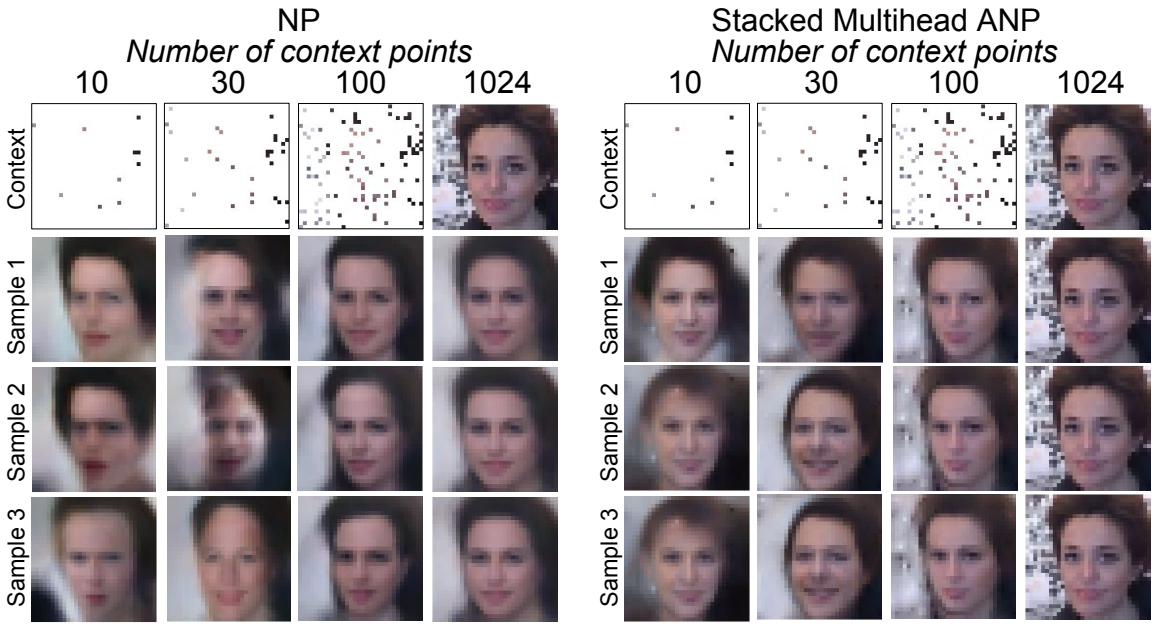


Figure 6.18: Same as Figure 6.4a but for a different image.

Figure 6.19: Mapping from 32×32 to 256×256 for different images.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Attentive Neural Processes
Publication Status	□Published x Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, M., Vinyals, O. and Teh, Y.W., 2019, May. Attentive Neural Processes. In Proceedings of the International Conference on Learning Representations (ICLR).

Student Confirmation

Student Name:	Hyun Jik Kim		
Contribution to the Paper	- Solo lead of the project, with collaborators (who implemented and executed the supplementary Bayesian Optimisation experiments and made the model figure) and advisors - Initiated and spearheaded the project by identifying the problem with previous work on Neural Processes (NPs) and came up with the core idea of using attention to resolve the issue - Wrote code and implemented experiments for all experiments of the project apart from the Bayesian Optimisation experiments in the appendix. This includes experiments for 1D GP regression, 2D image inpainting, bottom-half prediction and mapping between arbitrary resolutions. - All of the paper writing (with feedback from advisors).		
Signature		Date	

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Yee Whye Teh, Professor of Statistical Machine Learning		
Supervisor comments		
Signature	Date	

This completed form should be included in the thesis, at the end of the relevant chapter.

Chapter 7

Conclusion

In this thesis, we have explored contributions that lie at the intersection of probabilistic machine learning and interpretable machine learning. In Chapter 3, we have introduced the Tucker Gaussian Process (TGP), a regression model that regularises a GP towards more simpler regression functions, in particular a linear combination of separable functions. The linearity of the regression function further adds to the interpretability of TGP, since we may visualise the contributions of different additive components of the TGP to determine each of their roles. When TGP is applied to the task of collaborative filtering (CF), a domain where outputs can be effectively modelled as a linear combination of separable functions, we have shown that it is highly effective, and is in fact a generalisation of existing probabilistic matrix factorisation methods. We have also shown that it can naturally incorporate user/item side-information for enhancing the prediction of ratings via the kernel of the GP, that can encode similarities between different users and items. In doing so, we have bridged the gap between matrix factorisation methods and GP methods in CF.

In Chapter 4, we have proposed Scalable Kernel Composition (SKC), an algorithm that searches the space of interpretable GP models that can be translated into natural language. In particular, we have derived a cheap novel upper bound to the GP marginal likelihood that sandwiches the true marginal likelihood with the variational lower bound [Titsias, 2009]. In doing so, we have allowed SKC to scale beyond its predecessor CKS [Duvenaud et al., 2013] to bigger datasets, by reducing the algorithmic time complexity from $O(N^3)$ to $O(N^2)$. Equipped with this upper bound, SKC can discover global/local periodicities and linear trends in time series with tens of thousands of data points, that is well beyond the reach of CKS.

In Chapter 5, we have described FactorVAE, a method for learning independent factors

of variation in image datasets, that achieves higher disentanglement scores for the same reconstruction error than β-VAE [Higgins et al., 2016] on various benchmark datasets. The learned factors are quantified as the values of the latent dimensions of the VAE model, thus the method gives an interpretable representation of the factors of each image. The model also allows for post-hoc visualisation of each latent dimension by visualising latent traversals through the lens of the learned VAE decoder. Moreover, Euclidean distance in the learned latent space provide a more semantically meaningful notion of distance than Euclidean distance in the pixel space, adding to the interpretability of the method. Note that Euclidean distance within a latent dimension has a clear semantic meaning for disentangled latent representations, whereas this is not necessarily true for standard dimensionality reduction techniques. Furthermore, we have proposed a new disentanglement metric that is conceptually simpler, is free of hyperparameters, and avoids the failure mode of a former metric in Higgins et al. [2016]. With the high rise in interest in disentangling research since the publication of the results in this chapter, there have been a plethora of works that not only propose new methods that overcome the limitations of standard disentangling methods [Dupont, 2018, Esmaeili et al., 2018], but also analyse baseline approaches both empirically [Locatello et al., 2018] and mathematically [Rolinek et al., 2019]. It has been made clear in Locatello et al. [2018] that robust disentangling is difficult, and using supervised disentangling metrics for hyperparameter selection limits the validity of unsupervised approaches. This calls for methods with improved robustness as well as unsupervised metrics to guide model selection without requiring access to ground truth factors. Although Rolinek et al. [2019] has made a step towards understanding why these VAE-based disentangling methods are able to somewhat alleviate the identifiability issue outlined by Locatello et al. [2018], yet there remains a lot of work for a better theoretical understanding of why these methods are able to disentangle.

In Chapter 6, we have introduced Attentive Neural Processes (ANPs), a model for learning stochastic processes in a data-driven fashion, that augment Neural Processes (NPs) [Garnelo et al., 2018b] with attention to resolve the problem of underfitting. We have shown that this brings a noticeable improvement in predictive accuracy in terms of log likelihood, allows the model to be trained faster, and expands the range of functions that can be modelled. The mechanism of attention is interpretable in that it admits an intuitive explanation: it explicitly identifies the context points that are relevant for a given target prediction, allowing the decoder to concentrate on these points. In the case of multihead attention, the model's interpretability is enhanced as

we can look at the attention maps of each head, helping us understand the qualitative difference in the roles of each. The quantification of predictive uncertainty in the model also adds to the algorithmic transparency of the model, giving us an idea of when the model is certain or uncertain about its predictions.

Reverting the discussion to desirable properties of interpretable models outlines in Section 2.3, the challenges for present-day machine learning models lie in their lack of *transparency*, especially those that involve deep neural networks. Models optimised for performance, usually defined to be predictive accuracy in a discriminative task, usually have complicated computation that hurts *simulatability* (whether the computations involved are easy to understand for the user) and *decomposability* (whether each part of the model admits an intuitive explanation). These drawbacks are somewhat addressed by model designs that are true to their motivations, especially if their mechanisms are thoroughly tested via ablation studies, characteristic of Chapters 5 and 6. However it appears that the main difficulty that machine learning models face regarding interpretability is *algorithmic transparency*, guarantees that the model will or will not work in a given setting at test time. In this respect, the strand of work on verification of neural networks Dvijotham et al. [2018], that show provable guarantees on outputs for all possible inputs, appears to be a promising direction.

Bibliography

A. Achille and S. Soatto. Information Dropout: Learning optimal representations through noisy computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

D. Agarwal and B. Chen. Regression-based latent factor models. In *ACM SIGKDD*, 2009.

Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012.

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

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

M. A. Arcones and E. Gine. On the bootstrap of U and V statistics. *The Annals of Statistics*, pages 655–674, 1992.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein Generative Adversarial Networks. In *ICML*, 2017.

M. Aubry, D. Maturana, A. A. Efros, B. C. Russell, and J. Sivic. Seeing 3D chairs: exemplar part-based 2D-3D alignment using a large dataset of CAD models. In *CVPR*, 2014.

J. L. Ba, J. R. Kiros, and G. E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *ICML*, 2004.

Philip Bachman, Riashat Islam, Alessandro Sordoni, and Zafarali Ahmed. Vfunc: a deep generative model for functions. *arXiv preprint arXiv:1807.04106*, 2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Rémy Bardenet and Michalis Titsias. Inference for determinantal point processes without spectral knowledge. *NIPS*, 2015.

Sergey Bartunov and Dmitry P Vetrov. Fast adaptation in generative models with generative matching networks. In *AISTATS*, 2018.

Matthias Stephan Bauer, Mark van der Wilk, and Carl Edward Rasmussen. Understanding probabilistic sparse Gaussian process approximations. *NIPS*, 2016.

Matthew James Beal. *Variational algorithms for approximate Bayesian inference*. PhD thesis, University College London, 2003.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 35 (8):1798–1828, 2013.

G. Beylkin, J. Garcke, and M. Mohlenkamp. Multivariate regression and machine learning with sums of separable functions. *SIAM Journal on Scientific Computing*, 2009.

D. Billsus and M. Pazzani. Learning collaborative information filters. In *ICML*, 1998.

BODC. British Oceanographic Data Centre, UK Tide Gauge Network. `https://www.bodc.ac.uk/data/hosted_data_systems/sea_level/uk_tide_gauge_network/processed/`, 2017.

Edwin V Bonilla, Kian M Chai, and Christopher Williams. Multi-task Gaussian process prediction. In *NIPS*, 2008.

Jörg Bornschein, Andriy Mnih, Daniel Zoran, and Danilo Jimenez Rezende. Variational memory addressing in generative models. In *NIPS*, 2017.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

P. Brakel and Y. Bengio. Learning independent features with adversarial nets for non-linear ICA. *arXiv preprint arXiv:1710.05050*, 2017.

R. Bro. PARAFAC. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 1997.

Chris Burgess and Hyunjik Kim. 3d shapes dataset. https://github.com/deepmind/3dshapes-dataset/, 2018.

Tian Qi Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 2615–2625, 2018.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic Gradient Hamiltonian Monte Carlo. In *Proceedings of The 31st International Conference on Machine Learning*, 2014.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable representation learning by information maximizing Generative Adversarial Nets. In *NIPS*, 2016.

H. Chipman, E. George, and R. McCulloch. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 2010.

T. Cohen and M. Welling. Learning the irreducible representations of commutative lie groups. In *ICML*, 2014.

Corinna Cortes, Mehryar Mohri, and Ameet Talwalkar. On the impact of kernel approximation on learning accuracy. In *AISTATS*, 2010.

Kurt Cutajar, Michael A. Osborne, John P. Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. *ICML*, 2016.

Zhenwen Dai, Mauricio A Álvarez, and Neil Lawrence. Efficient modeling of latent information in supervised learning using Gaussian processes. In *NIPS*, 2017.

E. L. Denton and V. Birodkar. Unsupervised learning of disentangled representations from video. In *NIPS*, 2017.

G. Desjardins, A. Courville, and Y. Bengio. Disentangling factors of variation via generative entangling. *arXiv preprint arXiv:1210.5474*, 2012.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Been Doshi-Velez, Finale; Kim. Towards a rigorous science of interpretable machine learning. *eprint arXiv:1702.08608*, 2017.

Petros Drineas and Michael Mahoney. On the Nyström method for approximating a gram matrix for improved kernel-based learning. *JMLR*, 6:2153–2175, 2005.

S. Duane, A. Kennedy, B.Pendleton, and D.Roweth. Hybrid Monte Carlo. *Physics letters B*, 1987.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(Jul):2121–2159, 2011.

Emilien Dupont. Learning disentangled joint continuous and discrete representations. In *Advances in Neural Information Processing Systems*, pages 710–720, 2018.

David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *ICML*, 2013.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, pages 550–559, 2018.

C. Eastwood and C. Williams. A framework for the quantitative evaluation of disentangled representations. In *ICLR*, 2018.

Harrison Edwards and Amos Storkey. Towards a neural statistician. In *ICLR*, 2017.

R E Edwards. Positive definite functions and bochners theorem. In *Fourier Series*, pages 148–154. Springer, 1979.

SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

Babak Esmaeili, Hao Wu, Sarthak Jain, Alican Bozkurt, N Siddharth, Brooks Paige, Dana H Brooks, Jennifer Dy, and Jan-Willem van de Meent. Structured disentangled representations. *arXiv preprint arXiv:1804.02086*, 2018.

Reeves Fletcher and Colin M Reeves. Function minimization by conjugate gradients. *The computer journal*, 7(2):149–154, 1964.

Chris Fraley and Adrian E Raftery. Bayesian regularization for normal mixture estimation and model-based clustering. *Journal of classification*, 24(2):155–181, 2007.

Michael C Fu. Gradient estimation. *Handbooks in operations research and management science*, 13:575–616, 2006.

Jacob Gardner, Chuan Guo, Kilian Weinberger, Roman Garnett, and Roger Grosse. Discovering and exploiting additive structure for Bayesian optimization. In *AISTATS*, 2017.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *ICML*, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018b.

Carl-Friedrich Gauss. *Theoria combinationis observationum erroribus minimis obnoxiae*, volume 1. Henricus Dieterich, 1823.

A. Gelman and D. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 1992.

Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

C. W. Gini. Variability and mutability, contribution to the study of statistical distributions and relations. *Journal of American Statistical Association*, 66:534–544, 1971.

Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng. Measuring invariances in deep networks. In *NIPS*, 2009.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In *NIPS*, 2014a.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

R. Goroshin, J. Bruna, J. Tompson, D. Eigen, and Y. LeCun. Unsupervised learning of spatiotemporally coherent metrics. In *ICCV*, 2015.

Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Springer, 2012.

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. *arXiv preprint arXiv:1310.8499*, 2013.

Roger B. Grosse, Ruslan Salakhutdinov, William T. Freeman, and Joshua B. Tenenbaum. Exploiting compositionality to explore a large space of model structures. In *UAI*, 2012.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein GANs. In *NIPS*, 2017.

Isabelle Guyon, Imad Chaabane, Hugo Jair Escalante, Sergio Escalera, Damir Jajetic, James Robert Lloyd, Nria Maci, Bisakha Ray, Lukasz Romaszko, Michle Sebag, Alexander Statnikov, Sbastien Treguer, and Evelyne Viegas. A brief review of the chalearn automl challenge: Any-time any-dataset learning without human intervention. In *Proceedings of the Workshop on Automatic Machine Learning*, volume 64, pages 21–30, 2016.

P. Hall and C. Heyde. *Martingale Limit Theory and its Application*. Academic press, 2014.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *UAI*, 2013.

Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

Luke B Hewitt, Maxwell I Nye, Andreea Gane, Tommi Jaakkola, and Joshua B Tenenbaum. The variational homoencoder: Learning to learn high capacity generative models from few examples. In *UAI*, 2018.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. Beta-VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2016.

G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pages 44–51. Springer, 2011.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

M. D. Hoffman and M. J. Johnson. ELBO surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, 2016.

Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in Hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Tao Hong, Pierre Pinson, and Shu Fan. Global energy forecasting competition 2012, 2014.

W. N. Hsu, Y. Zhang, and J. Glass. Unsupervised learning of disentangled and interpretable representations from sequential data. In *NIPS*, 2017.

Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

M. Imaizumi and K. Hayashi. Doubly decomposing nonparametric tensor regression. In *ICML*, 2016.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Harold Jeffreys. Theory of probability, clarendon, 1961.

Chunlin Ji, Haige Shen, and Mike West. Bounded approximations for marginal likelihoods. 2010.

Alex Kendall and Yarin Gal. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.

Been Kim, Elena Glassman, Brittney Johnson, and Julie Shah. ibcm: Interactive Bayesian case model empowering humans via intuitive interaction. Technical report, MIT, 2015.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *NIPS*, 2014.

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

J. J. Kivinen and C. Williams. Transformation equivariant Boltzmann machines. In *International Conference on Artificial Neural Networks*, 2011.

Hermann König. *Eigenvalue distribution of compact operators*, volume 16. Birkhäuser, 2013.

Yehuda Koren. The bellkor solution to the Netflix grand prize. *Netflix prize documentation*, 81:1–10, 2009.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, pages 30–37, 2009.

T. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015.

A. Kumar, P. Sattigeri, and A. Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. In *ICLR*, 2018a.

Ananya Kumar, SM Eslami, Danilo J Rezende, Marta Garnelo, Fabio Viola, Edward Lockhart, and Murray Shanahan. Consistent generative query networks. *arXiv preprint arXiv:1807.02033*, 2018b.

B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pages 1–101, 2016.

N. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *NIPS*, 2004.

N. Lawrence and R. Urtasun. Non-linear matrix factorization with Gaussian processes. In *ICML*, 2009.

Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum Gaussian process regression. *JMLR*, 11: 1865–1881, 2010.

Judith Lean, Juerg Beer, and Raymond S Bradley. Reconstruction of solar irradiance since 1610: Implications for climate cbange. *Geophysical Research Letters*, 22(23), 1995.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *CVPR*, 2015.

Yingzhen Li and Richard E Turner. Rényi divergence variational inference. In *Advances in Neural Information Processing Systems*, pages 1073–1081, 2016.

Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3730–3738, 2015.

James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua Tenenbaum, and Zoubin Ghahramani. Automatic construction and natural-language description of nonparametric regression models. In *AAAI*, 2014.

Francesco Locatello, Stefan Bauer, Mario Lucic, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018.

Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. *arXiv preprint arXiv:1511.00830*, 2015.

Chao Ma, Yingzhen Li, and José Miguel Hernández-Lobato. Variational implicit processes. *arXiv preprint arXiv:1806.02390*, 2018.

Y. Ma, T. Chen, and E. Fox. A complete recipe for stochastic gradient MCMC. In *NIPS*, 2015.

David JC MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

A. Makhzani and B. Frey. PixelGAN autoencoders. In *NIPS*, 2017.

A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Gustavo Malkomes, Charles Schaff, and Roman Garnett. Bayesian optimization for automated model selection. In *NIPS*, 2016.

M. F. Mathieu, J. J. Zhao, A. Ramesh, P. Sprechmann, and Y. LeCun. Disentangling factors of variation in deep representation using adversarial training. In *NIPS*, 2016.

Alexander G de G Matthews, James Hensman, Richard E Turner, and Zoubin Ghahramani. On sparse variational methods and the Kullback-Leibler divergence between stochastic processes. *AISTATS*, 2016.

L. Matthey, I. Higgins, D. Hassabis, and A. Lerchner. dSprites: Disentanglement testing Sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3061–3070, 2015.

L. Mescheder, S. Nowozin, and A. Geiger. Adversarial variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks. In *ICML*, 2017.

Sean P Meyn and Richard L Tweedie. *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

Tom Minka. Divergence measures and message passing. Technical report, Microsoft Research, 2005.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018.

A. Mnih and D. J. Rezende. Variational inference for Monte Carlo objectives. In *ICML*, 2016a.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

Andriy Mnih and Danilo J Rezende. Variational inference for monte carlo objectives. *arXiv preprint arXiv:1602.06725*, 2016b.

John Morrissey, James L Sumich, and Deanna R. Pinkard-Meier. *Introduction To The Biology Of Marine Life*. Jones & Bartlett Learning, 1996.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT Press, Cambridge, MA, 2012a.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012b.

Radford M Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.

Radford M Neal and Geoffrey E Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

X. Nguyen, M. J. Wainwright, and M. I. Jordan. Estimating divergence functionals and the likelihood ratio by convex risk minimization. *IEEE Transactions on Information Theory*, 2010.

S. Nowozin, B. Cseke, and R. Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. In *NIPS*, 2016.

Junier B Oliva, Avinava Dubey, Andrew G Wilson, Barnabás Póczos, Jeff Schneider, and Eric P Xing. Bayesian nonparametric kernel-learning. In *AISTATS*, 2016.

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

S. Park, Y. Kim, and S. Choi. Hierarchical Bayesian matrix factorization with side information. In *IJCAI*, 2013.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, and Alexander Ku. Image transformer. In *ICML*, 2018.

P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter. A 3D face model for pose and illumination invariant face recognition. In *Proceedings of the IEEE International Conference on Advanced Video and Signal based Surveillance*, pages 296–301, 2009.

G. Perry, E. T. Rolls, and S. M. Stringer. Continuous transformation learning of translation invariant representations. *Experimental Brain Research*, 204(2):255–270, 2010.

Martin Piotte and Martin Chabbert. The pragmatic theory solution to the Netflix grand prize. *Netflix prize documentation*, 2009.

I. Porteous and M. Welling. Bayesian matrix factorization with side information and Dirichlet process mixtures. In *AAAI*, 2010.

Joaquin Quinonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *JMLR*, 6:1939–1959, 2005.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822, 2014.

Pramila Rani, Changchun Liu, Nilanjan Sarkar, and Eric Vanman. An empirical study of machine learning techniques for affect recognition in human–robot interaction. *Pattern Analysis and Applications*, 9(1):58–69, 2006.

C. Rasmussen and H. Nickisch. Gaussian Processes for Machine Learning (GPML) toolbox, 2010.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

S. Reed, K. Sohn, Y. Zhang, and H. Lee. Learning to disentangle factors of variation with manifold interaction. In *ICML*, 2014.

Scott Reed, Yutian Chen, Thomas Paine, Aäron van den Oord, SM Eslami, Danilo Rezende, Oriol Vinyals, and Nando de Freitas. Few-shot autoregressive density estimation: Towards learning to learn distributions. *arXiv preprint arXiv:1710.10304*, 2017.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Danilo Jimenez Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. One-shot generalization in deep generative models. *arXiv preprint arXiv:1603.05106*, 2016.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

Michal Rolinek, Dominik Zietlow, and Georg Martius. Variational autoencoders pursue pca directions (by accident). In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12406–12415, 2019.

Mihaela Rosca, Balaji Lakshminarayanan, and Shakir Mohamed. Distribution matching in variational inference. *arXiv preprint arXiv:1802.06847*, 2018.

Dan Rosenbaum, Frederic Besse, Fabio Viola, Danilo J Rezende, and SM Eslami. Learning models for visual 3d localization with implicit mapping. *arXiv preprint arXiv:1807.03149*, 2018.

Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Less is more: Nyström computational regularization. In *NIPS*, 2015.

W. Rudin. Fourier analysis on groups. *AMS*, 1964.

Y. Saatçi. *Scalable Inference for Structured Gaussian Process Models*. PhD thesis, University of Cambridge, 2011.

R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *NIPS*, 2008a.

R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using Markov Chain Monte Carlo. In *ICML*, 2008b.

Yves-Laurent Kom Samo and Stephen Roberts. Generalized spectral kernels. *arXiv preprint arXiv:1506.02236*, 2015.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge regression learning algorithm in dual variables. 1998.

J. Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879, 1992.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

Matthias Seeger, Christopher Williams, and Neil Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *AISTATS*, 2003.

Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

N. Siddharth, B. Paige, J. W. Van de Meent, A. Desmaison, F. Wood, N. D. Goodman, P. Kohli, and P. H. S. Torr. Learning disentangled representations with semi-supervised deep generative models. In *NIPS*, 2017.

A. Singh and G. Gordon. Relational learning via collective matrix factorization. In *ACM SIGKDD*, 2008.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NIPS*, 2017.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *NIPS*, 2005.

Arno Solin and Simo Särkkä. Explicit link between periodic covariance functions and state space models. *JMLR*, 2014.

C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár. Amortised MAP inference for image super-resolution. In *ICLR*, 2016.

Stan Development Team. Stan: A C++ library for probability and sampling, version 1.0, 2012.

M. Sugiyama, T. Suzuki, and T. Kanamori. Density-ratio matching under the Bregman divergence: a unified framework of density-ratio estimation. *Annals of the Institute of Statistical Mathematics*, 64(5):1009–1044, 2012.

Dougal J Sutherland and Jeff Schneider. On the error of random fourier features. *arXiv preprint arXiv:1506.02785*, 2015.

T. Suzuki, H. Kanagawa, H. Kobayashi, N. Shimizu, and Y. Tagami. Minimax optimal alternating minimization for kernel nonparametric tensor learning. In *NIPS*, 2016.

James Joseph Sylvester. Xxxvii. on the relation between the minor determinants of linearly equivalent quadratic functions. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1(4):295–305, 1851.

Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. Deep mixtures of factor analysers. *arXiv preprint arXiv:1206.4635*, 2012.

Pieter Tans and Ralph Keeling. NOAA/ESRL, Scripps Institution of Oceanography. `ftp://ftp.cmdl.noaa.gov/ccg/co2/trends/co2_mm_mlo.txt`, 2005.

Yee Whye Teh, Matthias Seeger, and Michael Jordan. Semiparametric latent factor models. In *AISTATS*, 2005.

Luke Tierney et al. A note on metropolis-hastings kernels for general state spaces. *The Annals of Applied Probability*, 8(1):1–9, 1998.

Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational Bayes for non-conjugate inference. In *International Conference on Machine Learning*, pages 1971–1979, 2014.

Michalis K Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *AISTATS*, 2009.

Andreas Töscher, Michael Jahrer, and Robert M Bell. The bigchaos solution to the Netflix grand prize. *Netflix prize documentation*, pages 1–52, 2009.

L. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 1966.

Pınar Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014. `http://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant`.

Jarno Vanhatalo, Jaakko Riihimäki, Jouni Hartikainen, Pasi Jylänki, Ville Tolvanen, and Aki Vehtari. Gpstuff: Bayesian modeling with Gaussian processes. *JMLR*, 14 (Apr):1175–1179, 2013.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NIPS*, 2016.

S. Watanabe. Information theoretical analysis of multivariate correlation. *IBM Journal of research and development*, 4(1):66–82, 1960.

K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *ICML*, 2009.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, 2011.

Christopher Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *NIPS*, 2001.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process kernels for pattern discovery and extrapolation. *arXiv preprint arXiv:1302.4245*, 2013.

Andrew Gordon Wilson, Christoph Dann, and Hannes Nickisch. Thoughts on massively scalable Gaussian processes. *arXiv preprint arXiv:1511.01870*, 2015. `http://arxiv.org/abs/1511.01870`.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *AISTATS*, 2016.

Max A Woodbury. Inverting modified matrices. *Memorandum report*, 42(106):336, 1950.

Z. Xu, F. Yan, and Y. Qi. Infinite Tucker decomposition: Nonparametric Bayesian models for multiway data analysis. In *ICML*, 2012.

H. H. Yang and S. I. Amari. Adaptive online learning algorithms for blind separation: maximum entropy and minimum mutual information. *Neural computation*, 9(7): 1457–1482, 1997.

I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998. `https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength`.

K. Yu, W. Chu, S. Yu, V. Tresp, and Z. Xu. Stochastic relational models for discriminative link prediction. In *NIPS*, 2006.

Kai Yu, Liang Ji, and Xuegong Zhang. Kernel nearest-neighbor algorithm. *Neural Processing Letters*, 15(2):147–156, 2002.

Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *International Conference on Machine Learning*, pages 325–333, 2013.

Q. Zhao, L. Zhang, and A. Cichocki. A tensor-variate Gaussian process for classification of multidimensional structured data. In *AAAI*, 2013.

S. Zhe, Y. Qi, Y. Park, Z. Xu, I. Molloy, and S. Chari. Dintucker: Scaling up Gaussian process models on large multidimensional arrays. In *AAAI*, 2016a.

S. Zhe, K. Zhang, P. Wang, K. Lee, Z. Xu, Y. Qi, and Z. Ghahramani. Distributed flexible nonlinear tensor factorization. In *NIPS*, 2016b.