Fast and Scalable Factor Analysis Algorithms for Bayesian Deep Learning

Scott Brownlie

Master of Science
School of Informatics
University of Edinburgh
2022

Abstract

This skeleton demonstrates how to use the `infthesis` style for MSc dissertations in Artificial Intelligence, Cognitive Science, Computer Science, Data Science, and Informatics. It also emphasises the page limit, and that you must not deviate from the required style. The file `skeleton.tex` generates this document and can be used as a starting point for your thesis. The abstract should summarise your report and fit in the space on the first page.

Acknowledgements

Any acknowledgements go here.

Table of Contents

1 Introduction **1**

2 Background **3**

2.1 Bayesian Linear Regression 3

2.2 Neural Networks 4

2.3 Stochastic Gradient Descent 5

2.4 Factor Analysis 6

2.5 Variational Inference 7

3 Related Work and Project Goals **8**

3.1 Related Work 8

3.1.1 Stochastic Weight Averaging 8

3.1.2 Stochastic, Low-rank Approximate Natural Gradient 9

3.2 Project Goals 11

4 Online Factor Analysis **12**

4.1 Online Stochastic Gradient Ascent 12

4.1.1 Partial derivatives with respect to $\mathbf{F}$ 13

4.1.2 Partial derivatives with respect to Ψ 14

4.1.3 Practical implementation 15

4.2 Online Expectation-Maximisation 16

4.2.1 E-step 18

4.2.2 M-step 18

4.2.3 Practical implementation 19

4.3 Factors Initialisation 21

4.4 Experiments 21

4.4.1 Results and Discussion 22

5 Variational Inference with a Factor Analysis Posterior 25
5.1 VIFA . . . 25
5.1.1 Partial Derivatives of the Variational Distribution . . . 25
5.1.2 Partial Derivatives of the Prior . . . 26
5.1.3 Partial Derivatives of the Likelihood . . . 27
5.1.4 Practical Implementation . . . 28
5.2 Experiments . . . 29
5.2.1 Posterior Estimation with Synthetic Data . . . 29
5.2.2 Posterior Estimation with UCI Datasets . . . 31
5.2.3 Neural Network Predictions . . . 36

6 Conclusions 39

Bibliography 41

A Derivations 44
A.1 Bayesian Linear Regression Posterior . . . 44

B Supplementary Results 46
B.1 Online Factor Analysis . . . 46
B.2 SGD as a Posterior Sampler . . . 47
B.2.1 Results and Discussion . . . 50
B.3 UCI Posterior Estimation . . . 54
B.4 UCI Neural Network Predictions . . . 57

Chapter 1

Introduction

In recent years, deep learning [6] has come to dominate many areas of machine learning research. Deep learning models continue to achieve state of the art results in several domains, such as CoAtNet-7 [4] for image classification and Megatron-LM [27] for language modelling. One reason for their success is the availability of huge training sets, such as the 300 millions images in JFT-300M [28] or the 100 million tokens in WikiText-103 [19]. The number of training examples, however, pales in comparison to the number of learnable parameters in some deep models. CoAtNet-7, for example, has 2.4 billion parameters, whereas Megatron-LM has a whopping 8.3 billion. This is a common feature of deep learning: the number of model parameters is often greater than the number of training examples. In this scenario, the model is severely *underspecified* by the data and many different settings of the parameters are able to explain the training set equally well. In some domains, choosing a single setting of the parameters and predicting a point estimate for each test example is satisfactory. However, for other critical applications, some measure of the uncertainty in the prediction is also required. This is where Bayesian deep learning comes in. Instead of choosing a single parameter vector $\theta \in \mathbb{R}^D$ for the neural network, predictions are made using *all* possible parameter vectors weighted by their posterior probabilities, given the training data $\mathcal{D}$. Formally, the posterior predictive distribution of the output y given the input $\mathbf{x}$ is

$$p(y|\mathbf{x}, \mathcal{D}) = \int_\theta p(y|\mathbf{x}, \theta) p(\theta|\mathcal{D}). \tag{1.1}$$

In practice, computing this integral is intractable due to the dimensionality and nonlinearities of a deep neural network. Alternatively, the integral can be approximated by a *Bayesian model average*,

$$p(y|\mathbf{x}, \mathcal{D}) \approx \frac{1}{L} \sum_{l=1}^{L} p(y|\mathbf{x}, \theta_l), \quad \theta_l \sim p(\theta|\mathcal{D}). \tag{1.2}$$

Hence, the focus of Bayesian deep learning is on estimating the *posterior*

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}, \tag{1.3}$$

where $p(\mathcal{D}|\theta)$ is the *likelihood* that θ generated $\mathcal{D}$, $p(\theta)$ is the *prior* and $p(\mathcal{D})$ is the *marginal likelihood.* Unfortunately, this calculation is also intractable due to the high-dimensional integral $p(\mathcal{D}) = \int_{\theta} p(\mathcal{D}|\theta)p(\theta)$. A common alternative is to approximate the posterior with a Gaussian distribution with mean $\mu \in \mathbb{R}^D$ and covariance $\Sigma \in \mathbb{R}^{D \times D}$, denoted by $\mathcal{N}(\mu, \Sigma)$. The key practical challenge is then finding a way to approximate Σ, since the full $D \times D$ covariance matrix will not fit into memory for large D.

One of the simplest approaches is to use a diagonal - or *mean-field* - approximation [2, 8, 9, 12, 24]. This is convenient in the sense that the covariance matrix is completely specified by D parameters, but it does not allow for any posterior correlations between different elements of θ. A more flexible approach is a low-rank plus diagonal approximation, $\mathbf{F}\mathbf{F}^\intercal + \Psi$, where $\mathbf{F} \in \mathbb{R}^{D \times K}$ and Ψ is a $D \times D$ diagonal matrix. This preserves some of the off-diagonal structure of the covariance matrix and is practically feasible for $K \ll D$. Two recent methods which adopt this approach are SWAG [17] and SLANG [20]. While both algorithms have achieved promising results, they also have fundamental limitations. In the case of SWAG, it does not make use of all the available data to fit the low-rank plus diagonal covariance matrix, whereas SLANG can only be applied to standard feed-forward neural networks.

The work in this thesis is based on the low-rank plus diagonal factor analysis model, which can help to overcome some of the limitations of both SWAG and SLANG. The main contributions are:

- two novel online algorithms for learning factor analysis models of high-dimensional data, which can be used in conjunction with SWAG;
- a novel variational inference algorithm similar to SLANG, but which can be applied to any neural network architecture.

Chapter 2 introduces the background required to follow the rest of the thesis. Chapter 3 describes SWAG and SLANG in more detail and sets out the main goals that the thesis hopes to achieve. Chapter 4 presents the online factor analysis algorithms and the results of applying them to synthetic datasets. Chapter 5 introduces VIFA, a novel variational inference algorithm, and demonstrates its effectiveness in estimating the posterior and making predictions. Finally, Chapter 6 brings the thesis to an end with some conclusions and suggestions for future work.

Chapter 2

Background

This chapter presents the background necessary to understand the rest of the thesis. Bayesian linear regression, whose posterior can be computed in closed for, provides a nice testbed for the algorithms. Moreover, it can be viewed as a simple neural network with a single affine layer from input to output. Therefore, it is introduced first, followed by the general case of a neural network with multiple layers. The most common training algorithm for neural networks, stochastic gradient descent, is described next. Factor analysis is then introduced, as this is the basis of the main contributions in this thesis. Finally, variational inference is described as a pre-requisite for algorithms which use this approach to learn the posterior distribution of a neural network.

2.1 Bayesian Linear Regression

A linear regression model is a mapping from vector inputs $\mathbf{x} \in \mathbb{R}^D$ to scalar outputs $y \in \mathbb{R}$ via an affine transformation. Given a set of observed input-output pairs, $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, it is assumed that each output is generated according to

$$y_n = \theta^\intercal \mathbf{x}_n + \varepsilon \tag{2.1}$$

for some unknown $\theta \in \mathbb{R}^D$. The underlying signal, $\theta^\intercal \mathbf{x}_n$, is corrupted by additive noise,

$$\varepsilon \sim \mathcal{N}(0, \sigma^2), \tag{2.2}$$

for some $\sigma > 0$ [1]. The model is often written with an explicit bias term, but this can be absorbed into θ by adding a constant of one to the input, leading to the expression in Equation (2.1). Due to the additive noise, each y_n is a random variable, conditioned on $\mathbf{x}_n$ and θ. Since ε is Gaussian distributed, the conditional pdf of y_n is

$$p(y_n|\theta, \mathbf{x}_n) = \mathcal{N}\big(\theta^\mathsf{T}\mathbf{x}_n, \sigma^2\big) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}\big(y_n - \theta^\mathsf{T}\mathbf{x}_n\big)^2\Big). \tag{2.3}$$

Assuming that the observations in $\mathcal{D}$ are independent and identically distributed (iid), the log-likelihood of θ having generated the data is

$$\begin{aligned}\log p(\mathcal{D}|\theta) &= \sum_{n=1}^{N}\big[\log p(y_n|\theta,\mathbf{x}_n) + \log p(\mathbf{x}_n)\big]\\ &= \sum_{n=1}^{N}\Big[-\frac{1}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\big(y_n - \theta^\mathsf{T}\mathbf{x}_n\big)^2\Big] + \sum_{n=1}^{N}\log p(\mathbf{x}_n)\\ &= -\frac{1}{2\sigma^2}\sum_{n=1}^{N}\big(y_n - \theta^\mathsf{T}\mathbf{x}_n\big)^2 - \frac{N}{2}\log\sigma^2 - \frac{N}{2}\log 2\pi + \sum_{n=1}^{N}\log p(\mathbf{x}_n)\\ &= -\frac{\beta}{2}\sum_{n=1}^{N}\big(y_n - \theta^\mathsf{T}\mathbf{x}_n\big)^2 + \frac{N}{2}\log\beta + \text{constant},\end{aligned} \tag{2.4}$$

where $\beta = \frac{1}{\sigma^2}$ is the *noise precision* [1]. In Bayesian linear regression, a prior distribution for θ is also specified. A common choice is

$$p(\theta) = \mathcal{N}\big(\mathbf{0}, \alpha^{-1}\mathbf{I}\big) = \Big(\frac{\alpha}{2\pi}\Big)^{\frac{D}{2}} \exp\Big(-\frac{\alpha}{2}\theta^\mathsf{T}\theta\Big) \tag{2.5}$$

for some $\alpha > 0$, which is a hyperparameter known as the *prior precision* [1]. Given the log-likelihood and the prior, it turns out that the posterior distribution of θ is Gaussian and can be computed in closed form. Formally,

$$p(\theta|\mathcal{D}) = \mathcal{N}(\mathbf{A}^{-1}\mathbf{b}, \mathbf{A}^{-1}), \tag{2.6}$$

where

$$\mathbf{A} = \alpha\mathbf{I} + \beta\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^\mathsf{T} \quad \text{and} \quad \mathbf{b} = \beta\sum_{n=1}^{N} y_n\mathbf{x}_n. \tag{2.7}$$

See Appendix A.1 for a full derivation.

2.2 Neural Networks

Let $f(\mathbf{x}) : \mathbb{R}^J \mapsto \mathbb{R}$ be a neural network with a single *hidden* layer. Its functional from can be written as

$$f(\mathbf{x}) = b + \sum_{h=1}^{H} v_h \cdot g(\mathbf{w}_h^\mathsf{T}\mathbf{x} + b_h), \tag{2.8}$$

where H is the number of *units* in the hidden layer, the weights $\mathbf{w}_h \in \mathbb{R}^J$, $v_h \in \mathbb{R}$ and biases $b_h, b \in \mathbb{R}$ are learnable parameters and $g(x) : \mathbb{R} \mapsto \mathbb{R}$ is a non-linear *activation* function [6]. Notice that this is similar to the functional form of linear regression. The difference is that, instead of the output being a linear combination of the elements of $\mathbf{x}$, it is a linear combination of the $g(\mathbf{w}_h^\intercal \mathbf{x} + b_h)$, which can be viewed as higher-order features of the input. Even higher-order features can be used by adding more layers. For example, the functional form of a neural network with two hidden layers can be written in matrix form as

$$f(\mathbf{x}) = \mathbf{w}_2^T \Big(g_2\big(\mathbf{W}_1 g_1(\mathbf{W}_0 \mathbf{x} + \mathbf{b}_0) + \mathbf{b}_1\big) \Big) + b_2, \tag{2.9}$$

where the weights $\mathbf{W}_0 \in \mathbb{R}^{H_0 \times J}$, $\mathbf{W}_1 \in \mathbb{R}^{H_1 \times H_0}$, $\mathbf{w}_2 \in \mathbb{R}^{H_1}$ and biases $\mathbf{b}_0 \in \mathbb{R}^{H_0}$, $\mathbf{b}_1 \in \mathbb{R}^{H_1}$, $b_2 \in \mathbb{R}$ are learnable parameters and $g_1(x) : \mathbb{R} \mapsto \mathbb{R}$, $g_2(x) : \mathbb{R} \mapsto \mathbb{R}$ are non-linear activation functions, applied element-wise to the vectors. The total number of learnable parameters in this model is $D = H_0(J+1) + H_1(H_0+1) + (H_1+1)$, and they are collectively denoted by $\theta \in \mathbb{R}^D$. This network is already considered *deep*, as it has multiple hidden layers. However, it is not uncommon for neural networks to have tens or even hundreds of hidden layers and millions of learnable parameters [4, 27]. Unlike in linear regression, it is not possible to compute the posterior of θ in closed due to the non-linear activation function(s). Instead, numerical methods are required. Moreover, due to the large dimensionality of θ, it is usually impractical to estimate the full $D \times D$ posterior covariance matrix and some simplification must be made.

2.3 Stochastic Gradient Descent

The parameters θ of a neural network f_θ are most commonly learned via stochastic gradient descent (SGD) [6]. Given training data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, this iterative algorithm usually begins from a random initialisation of θ and then proceeds to update θ in the direction of the negative gradient of the training loss. Typical loss functions are mean squared error for regression and cross-entropy loss for classification, and it is common to add a *regularisation* term to prevent overfitting. There are many variants of SGD [25], but the vanilla update rule on iteration t is

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \left(\frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},y) \in \mathcal{B}} \mathcal{L}_P\big(y, f_{\theta_t}(\mathbf{x})\big) + \lambda \mathcal{L}_R(\theta_t) \right), \tag{2.10}$$

where $\mathcal{B} \subset \mathcal{D}$ is a mini-batch of training examples, $\mathcal{L}_P\big(y, f_{\theta_t}(\mathbf{x})\big)$ is the prediction loss for training example $(\mathbf{x}, y)$ and $\lambda \mathcal{L}_R(\theta_t)$ is the regularisation penalty [3]. The regularisation strength $\lambda > 0$ and the learning rate $\eta > 0$ are hyperparameters. In each iteration, the network first predicts $\{f_{\theta_t}(\mathbf{x}) | (\mathbf{x}, y) \in \mathcal{B}\}$ in the *forward pass*. Next, the average mini-batch regularised training loss in computed. Then the partial derivatives $\nabla_{\theta_t}(\cdot)$ are obtained via the back-propagation algorithm [26], which uses the chain rule from calculus to *back-propagate* the gradients through the network. Finally, the gradient vector is used to update the parameters.

2.4 Factor Analysis

Factor analysis (FA) is a latent variable model which generates observations $\theta \in \mathbb{R}^D$ as follows. First, a latent vector $\mathbf{h} \in \mathbb{R}^K$, for some $K < D$, is sampled from $p(\mathbf{h}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. Next, $\mathbf{h}$ is transformed onto a K-dimensional linear subspace of $\mathbb{R}^D$ by left-multiplying it by a *factor loading* matrix $\mathbf{F} \in \mathbb{R}^{D \times K}$. The origin of this subspace is then shifted by adding a bias term $\mathbf{c} \in \mathbb{R}^D$. Finally, the data is perturbed by adding some zero mean Gaussian noise $\varepsilon \in \mathbb{R}^D$ sampled from $\mathcal{N}(\mathbf{0}, \Psi)$, where Ψ is a $D \times D$ diagonal matrix [1]. Putting all this together, an observation $\theta \in \mathbb{R}^D$ is generated according to

$$\theta = \mathbf{Fh} + \mathbf{c} + \varepsilon. \tag{2.11}$$

In the context of this thesis, an observation θ is the parameter vector of a neural network. It follows from Equation (2.11) that, given $\mathbf{h}$, the observations θ are Gaussian distributed with mean $\mathbf{Fh} + \mathbf{c}$ and covariance Ψ [1]. Formally,

$$p(\theta|\mathbf{h}) = \mathcal{N}\big(\mathbf{Fh} + \mathbf{c}, \Psi\big) = \frac{1}{\sqrt{(2\pi)^D |\Psi|}} \exp\Big(-\frac{1}{2}(\theta - \mathbf{Fh} - \mathbf{c})^\intercal \Psi^{-1} (\theta - \mathbf{Fh} - \mathbf{c})\Big), \tag{2.12}$$

where $|\Psi|$ is the *determinant* of Ψ. From [1], integrating $p(\theta|\mathbf{h})$ over $\mathbf{h}$ gives the marginal distribution

$$p(\theta) = \mathcal{N}\big(\mathbf{c}, \mathbf{FF}^\intercal + \Psi\big). \tag{2.13}$$

Given only observations $\theta_1, \theta_2, \dots, \theta_T$, a model of this form can be fit to the data. The parameters of the model are $\mathbf{c}, \mathbf{F}$ and Ψ. The value of $\mathbf{c}$ which maximises the likelihood of the observed data is the empirical mean of the observations [1]. Having set $\mathbf{c}$, an expectation-maximisation (EM) or singular value decomposition (SVD) algorithm can find the maximum likelihood estimates of $\mathbf{F}$ and Ψ [1]. However, both are batch methods which require storing all the observations in memory, making them impractical for high-dimensional data, such as the parameter vectors of deep neural networks. In this case, alternative online algorithms are needed.

2.5 Variational Inference

For linear regression, computing the posterior in Equation (2.6) is possible because the log-likelihood, $p(\mathcal{D}|\theta)$, is a quadratic expression of θ. However, when θ is the parameter vector of a neural network with even a single non-linear hidden layer, the log-likelihood is not so simple and the posterior cannot be written in closed-form. Instead, the marginal likelihood, $p(\mathcal{D}) = \int_\theta p(\mathcal{D}|\theta)p(\theta)$, must be evaluated, which is intractable for high-dimensional θ. An alternative strategy is to approximate the posterior with a Gaussian distribution $q(\theta) = \mathcal{N}(\mu, \Sigma)$, where the parameters μ and Σ are chosen to minimise some measure of dissimilarity between $q(\theta)$ and $p(\theta|\mathcal{D})$. Because $q(\theta)$ is essentially a variable in an optimisation procedure, this approach is known as *variational inference* (VI) [1]. A common measure of dissimilarity between two distributions is the Kullback-Leibler (KL) divergence [1], which in this case is

$$\begin{aligned} \mathbb{D}_{KL}[q(\theta)\|p(\theta|\mathcal{D})] &= \mathbb{E}_{q(\theta)}\Big[\log\frac{q(\theta)}{p(\theta|\mathcal{D})}\Big] \\ &= \mathbb{E}_{q(\theta)}[\log q(\theta) - \log p(\theta) - \log p(\mathcal{D}|\theta) + \log p(\mathcal{D})]. \end{aligned} \tag{2.14}$$

Since $p(\mathcal{D})$ is a constant, it can be ignored and the optimal distribution can be obtained by solving

$$\min_{\mu,\Sigma}\big[\mathbb{E}_{q(\theta)}[\log q(\theta)] - \mathbb{E}_{q(\theta)}[\log p(\theta)] - \mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)]\big]. \tag{2.15}$$

One way to tackle this is via an iterative gradient algorithm. Depending on the structure of Σ and the form of the prior $p(\theta)$, it may be possible to compute the partial derivatives $\nabla_{\mu,\Sigma}\mathbb{E}_{q(\theta)}[\log q(\theta)]$ and $\nabla_{\mu,\Sigma}\mathbb{E}_{q(\theta)}[\log p(\theta)]$ exactly [14]. However, when the likelihood is parameterised by a neural network, is not possible to obtain $\nabla_{\mu,\Sigma}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)]$ analytically. Alternatively, the true gradient can be approximated by Monte Carlo estimates of the form

$$\nabla_{\mu,\Sigma}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] \approx \nabla_{\mu,\Sigma}\left(\frac{1}{L}\sum_{l=1}^{L}\left(\frac{N}{M}\sum_{(\mathbf{x},y)\in\mathcal{B}_l}\log p(y|\mathbf{x},\theta_l)\right)\right), \tag{2.16}$$

where $\theta_l \sim \mathcal{N}(\mu,\Sigma)$ and each $\mathcal{B}_l$ is a mini-batch of M training examples sampled from the full dataset $\mathcal{D}$ of size N. Normally it would not be possible to compute these partial derivatives with respect to μ and Σ, given that they only take part via the sampling operation. However, this can in fact be done by using the *re-parameterisation trick* [6]. Instead of sampling θ_l from $\mathcal{N}(\mu,\Sigma)$ directly, a random variable $\mathbf{z}_l$ is sampled from $\mathcal{N}(\mathbf{0},\mathbf{I})$ and then transformed to θ_l via a deterministic operation involving μ and Σ. This simple trick means that Equation (2.16) can be evaluated and used in conjunction with SGD or any of its variants to optimise the objective in Equation (2.15).

Chapter 3

Related Work and Project Goals

This chapter presents the SWAG and SLANG methods. Both are promising approaches to Bayesian deep learning, but have fundamental limitations which are described in detail. The project goals, which focus on overcoming these limitations via FA, are then clearly set out.

3.1 Related Work

3.1.1 Stochastic Weight Averaging

Under certain assumptions on the learning rate, SGD is guaranteed to converge to a local minimum of the (regularised) training loss [25]. While this is a nice property, the training loss is only a proxy for finding solutions with low test error. Recent work suggests that good regions of the loss surface with respect to train and test error are rarely perfectly aligned [11]. Moreover, SGD with a decaying learning rate actually converges to solutions on the *periphery* of a set of parameters with low test error. It is thought that such solutions lie in sharp regions of the training loss surface, where even a small shift can result in poor generalisation [11]. Therefore, solutions in flatter regions of the training loss surface, which are more likely to remain good even are being shifted, are desirable. When used with a high constant learning rate, SGD first approaches a local minimum and then proceeds to bounce around in its vicinity [18]. In doing so, it is thought that SGD visits those points which are on the periphery of regions corresponding to low test error, while hopping back and forth over a flat region of the training loss surface [11]. By averaging the SGD iterates, a solution closer to the centre of this flat region can be obtained. This is the idea behind the SWA solution,

θ_{SWA}, from [11]. Formally, given parameter vectors $\theta_1, \theta_2, \ldots, \theta_T \in \mathbb{R}^D$ encountered along the SGD trajectory,

$$\theta_{\text{SWA}} = \frac{1}{T} \sum_{t=1}^{T} \theta_t. \tag{3.1}$$

There is compelling evidence that the SWA solution leads to better generalisation than the SGD solution, even though the SGD solution often achieves lower training error.

One limitation of the SWA solution is that it does not convey the uncertainty in the model parameters. The method was later extended to also include a low-rank plus diagonal approximation of the covariance matrix of the SGD iterates [17]. The diagonal part is

$$\frac{1}{2}\Sigma_{\text{diag}} = \frac{1}{2}\text{diag}\left(\frac{1}{T}\sum_{t=1}^{T} \theta_t^2 - \theta_{\text{SWA}}^2\right), \tag{3.2}$$

where the squares are applied element-wise. For rank $K \in \mathbb{N}^+$, the low-rank part is

$$\frac{1}{2(K-1)} \hat{\mathbf{D}}\hat{\mathbf{D}}^\mathsf{T}, \tag{3.3}$$

where $\hat{\mathbf{D}} \in \mathbb{R}^{D \times K}$ is formed by concatenating the vectors $(\theta_{t_1} - \overline{\theta}_{t_1}), \ldots, (\theta_{t_K} - \overline{\theta}_{t_K})$ for some $t_1, \ldots, t_K \in \{1, 2, \ldots, T\}$, where $\overline{\theta}_{t_i}$ is the running average of the parameter vector after iteration t_i. Note that

$$\frac{1}{K-1} \hat{\mathbf{D}}\hat{\mathbf{D}}^\mathsf{T} = \frac{1}{K-1} \sum_{i=1}^{K} (\theta_{t_i} - \overline{\theta}_{t_i})(\theta_{t_i} - \overline{\theta}_{t_i})^\mathsf{T}, \tag{3.4}$$

which is similar to the empirical covariance of $\theta_{t_1}, \theta_{t_2}, \ldots, \theta_{t_K}$, except that the running averages $\overline{\theta}_{t_i}$ are used instead of the true mean, which is not know during training. Together, the mean and the approximate covariance give rise to an approximate posterior distribution,

$$p(\theta|\mathcal{D}) \approx \mathcal{N}\left(\theta_{\text{SWA}}, \frac{1}{2(K-1)} \hat{\mathbf{D}}\hat{\mathbf{D}}^\mathsf{T} + \frac{1}{2}\Sigma_{\text{diag}}\right). \tag{3.5}$$

This method is known as SWA-Gaussian, or SWAG [17]. One undesirable property of SWAG is that the rank K and the number of SGD iterates used to construct $\hat{\mathbf{D}}$ are one and the same. This means that, if $K \ll T$, only a tiny subset of $\theta_1, \theta_2, \ldots, \theta_T$ will be used to estimate $\hat{\mathbf{D}}$ and the rest will be wasted. In SWAG, the authors construct $\hat{\mathbf{D}}$ using only the final parameter vector from each of the last K training epochs.

3.1.2 Stochastic, Low-rank Approximate Natural Gradient

The Variational Online Gauss-Newton (VOGN) method [12] is one approach to optimising the variational objective in Equation (2.15). It learns the parameters of the variational distribution, μ and Σ, via an approximate natural gradient algorithm. The online update rule is

$$\mu \leftarrow \mu - \eta_\mu \Sigma(\hat{\mathbf{g}}_\theta + \alpha\mu), \quad \Sigma^{-1} \leftarrow (1-\eta_\Sigma)\Sigma^{-1} + \eta_\Sigma(\hat{\mathbf{G}}_\theta + \alpha\mathbf{I}), \tag{3.6}$$

where $\eta_\mu, \eta_\Sigma > 0$ are learning rates, α is the precision of the prior and $\hat{\mathbf{g}}_\theta$ and $\hat{\mathbf{G}}_\theta$ are, respectively, a Monte Carlo estimate of the gradient of the negative log-likelihood and an Empirical Fisher matrix, given $\theta \sim q(\theta) = \mathcal{N}(\mu, \Sigma)$. Formally,

$$\hat{\mathbf{g}}_\theta = -\frac{N}{M} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{B}} \mathbf{g}_\theta^i, \quad \hat{\mathbf{G}}_\theta = -\frac{N}{M} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{B}} \mathbf{g}_\theta^i (\mathbf{g}_\theta^i)^\intercal, \tag{3.7}$$

where $\mathbf{g}_\theta^i = \nabla_\theta \log p(y_i|\mathbf{x}_i, \theta)$ and $\mathcal{B}$ is a mini-batch of M training examples sampled from the full dataset $\mathcal{D}$ of size N. Since θ is the parameter vector of a neural network, the gradients $\mathbf{g}_\theta^i$ can be computed via back-propagation [26]. However, for deep neural networks with D parameters, the update in Equation (3.6) is generally infeasible, as it involves inverting the $D \times D$ covariance matrix Σ.

A computationally tractable version of VOGN can be obtained by using a low-rank plus diagonal approximation of the inverse of the covariance matrix. This is the approach taken by SLANG [20], which stands for stochastic, low-rank, approximate natural gradient. This algorithm sets $\Sigma^{-1} = \mathbf{U}\mathbf{U}^\intercal + \mathbf{D}$ for some $\mathbf{U} \in \mathbb{R}^{D \times K}$ and a $D \times D$ diagonal matrix $\mathbf{D}$. Substituting this definition into Equation (3.6), the update can be written as

$$\Sigma^{-1} \leftarrow \left[(1-\eta_\Sigma)\mathbf{U}\mathbf{U}^\intercal + \eta_\Sigma \hat{\mathbf{G}}_\theta\right] + \left[(1-\eta_\Sigma)\mathbf{D} + \eta_\Sigma \alpha \mathbf{I}\right]. \tag{3.8}$$

To make this tractable, the non-diagonal component of the right-hand side is approximated as

$$(1-\eta_\Sigma)\mathbf{U}\mathbf{U}^\intercal + \eta_\Sigma \hat{\mathbf{G}}_\theta \approx \mathbf{Q}_{1:K} \Lambda_{1:K} \mathbf{Q}_{1:K}^\intercal, \tag{3.9}$$

where $\Lambda_{1:K}$ is a $K \times K$ diagonal matrix containing the K largest eigenvalues of $(1 - \eta_\Sigma)\mathbf{U}\mathbf{U}^\intercal + \eta_\Sigma \hat{\mathbf{G}}_\theta$, and the columns of $\mathbf{Q}_{1:K} \in \mathbb{R}^{D \times K}$ are the corresponding eigenvectors. In order to compute this eigen-decomposition, the individual gradients $\mathbf{g}_\theta^i$ for each $(\mathbf{x}_i, y_i) \in \mathcal{B}$ are required. When using the standard back-propagation algorithm in deep learning frameworks such as PyTorch [21], this is only possible by looping over the mini-batch samples one-by-one and storing each $\mathbf{g}_\theta^i$, which is very inefficient. A faster approach - which was adopted by SLANG - is outlined in [7] for standard neural networks. However, the authors of SLANG admit that this speed-up is limited to feed-forward layers and a separate implementation would be required to handle each different type of layer.

3.2 Project Goals

Although SWAG and SLANG are distinctly different approaches to Bayesian deep learning, they both opt for a low-rank plus diagonal approximation to simplify the estimation of the posterior covariance matrix. The low-rank plus diagonal covariance is also a defining feature of the FA model. However, there is more to the FA than that. In particular, it is a latent variable model which generates observations according to Equation (2.11). **The overall objective of this project is to use these specific properties of FA to overcome some of the limitations of SWAG and SLANG.**[1]

Online Factor Analysis. The first goal of this project is motivated by the fact that SWAG does not make use of all the available data when estimating the low-rank plus diagonal approximation of the posterior covariance. This is unnecessary, since a covariance matrix with the same structure can be learned from *all* the SGD iterates via FA. To see this, let $\mathbf{F} = \frac{1}{\sqrt{2(K-1)}}\hat{\mathbf{D}}$ and $\Psi = \frac{1}{2}\Sigma_{\text{diag}}$. Then the SWAG posterior from Equation (3.5) becomes $\mathcal{N}\big(\theta_{\text{SWA}}, \mathbf{F}\mathbf{F}^\intercal + \Psi\big)$, which is the same as the FA model from Equation (2.13) with $\mathbf{c} = \theta_{\text{SWA}}$ [3]. In theory, such a model could be fit to $\theta_1, \theta_2, \ldots, \theta_T$ using either the EM or SVD algorithm from [1]. However, given these are both batch algorithms, the SGD iterates would have to be stored in memory, which is impractical for deep neural networks with millions of parameters. Hence, an online, iterative approach is required. To the best of this author's knowledge, no such algorithm exists. **To fill this gap, this thesis presents two novel online FA algorithms which scale to high-dimensional data, such as the parameter vectors of deep neural networks.**

Variational Inference with a Factor Analysis Posterior. The second goal of this project is motivated by the fact that SLANG can only be applied to feed-forward neural networks. More complex architectures are extremely common in research and industry, especially convolutional layers [15] in computer vision and recurrent [10] and transformer [29] layers in natural language processing. In these domains, the applicability of SLANG is limited. **With a focus on practicality, this thesis presents a novel, model-agnostic, VI algorithm which learns a FA posterior of a neural network. This method sacrifices the natural gradient used by SLANG in favour of the standard gradient, such that it can be applied to any neural network architecture with no extra effort.**

[1] Code for all algorithms and experiments is available at `https://github.com/ayeright/swafa`

Chapter 4

Online Factor Analysis

This chapter presents two algorithms which can be used to approximate the posterior distribution of the parameter vector of a neural network with a FA model, assuming that samples of the posterior can be obtained via some separate procedure, such as that employed by SWA and its variants. Crucially, both algorithms are executed online in an iterative fashion, making them suitable for high-dimensional deep neural networks.

4.1 Online Stochastic Gradient Ascent

Suppose that samples from the posterior, $\theta_1, \theta_2, \ldots, \theta_T$, are readily available. In situations where learning latent variable models with the classic EM algorithm is slow, [1] suggests optimising the log-likelihood of the model parameters via gradient methods. Since FA is a latent variable model, this approach can be applied here. In this case, the log-likelihood of the parameters $\mathbf{F}$ and Ψ given the observed data is

$$L(\mathbf{F}, \Psi) = \frac{1}{T} \sum_{t=1}^{T} \log p(\theta_t | \mathbf{F}, \Psi). \tag{4.1}$$

The partial derivatives of the log-likelihood with respect to $\mathbf{F}$ and Ψ are therefore

$$\nabla_{\mathbf{F},\Psi} L(\mathbf{F}, \Psi) = \frac{1}{T} \sum_{t=1}^{T} \nabla_{\mathbf{F},\Psi} \log p(\theta_t | \mathbf{F}, \Psi). \tag{4.2}$$

Computing $\nabla_{\mathbf{F},\Psi} L(\mathbf{F}, \Psi)$ in full would require a pass over all observations, $\theta_1, \theta_2, \ldots, \theta_T$. However, a stochastic gradient algorithm can be used instead, as long as the expectation of the sample derivatives is proportional to $\nabla_{\mathbf{F},\Psi} L(\mathbf{F}, \Psi)$. By using the $\nabla_{\mathbf{F},\Psi} \log p(\theta_t | \mathbf{F}, \Psi)$ as the sample derivatives, this condition clearly holds. Hence, as long as the partial derivatives $\nabla_{\mathbf{F},\Psi} \log p(\theta_t | \mathbf{F}, \Psi)$ can be computed efficiently, they can be used in conjunction with SGD or any of its variants to optimise $\mathbf{F}$ and Ψ online [3].

By adapting an argument for general latent variable models in [1] to FA, the required sample derivatives can be written as

$$\begin{aligned}
\nabla_{\mathbf{F},\Psi} \log p(\theta_t|\mathbf{F},\Psi) &= \frac{1}{p(\theta_t|\mathbf{F},\Psi)} \nabla_{\mathbf{F},\Psi} p(\theta_t|\mathbf{F},\Psi) \\
&= \frac{1}{p(\theta_t|\mathbf{F},\Psi)} \int_{\mathbf{h}_t} \nabla_{\mathbf{F},\Psi} p(\theta_t,\mathbf{h}_t|\mathbf{F},\Psi) \\
&= \frac{1}{p(\theta_t|\mathbf{F},\Psi)} \int_{\mathbf{h}_t} p(\theta_t,\mathbf{h}_t|\mathbf{F},\Psi) \nabla_{\mathbf{F},\Psi} \log p(\theta_t,\mathbf{h}_t|\mathbf{F},\Psi) \\
&= \int_{\mathbf{h}_t} \frac{p(\theta_t,\mathbf{h}_t|\mathbf{F},\Psi)}{p(\theta_t|\mathbf{F},\Psi)} \nabla_{\mathbf{F},\Psi} \log p(\theta_t,\mathbf{h}_t|\mathbf{F},\Psi) \\
&= \int_{\mathbf{h}_t} p(\mathbf{h}_t|\theta_t,\mathbf{F},\Psi) \nabla_{\mathbf{F},\Psi} \log p(\theta_t,\mathbf{h}_t|\mathbf{F},\Psi) \\
&= \mathbb{E}_{p(\mathbf{h}_t|\theta_t,\mathbf{F},\Psi)} \left[\nabla_{\mathbf{F},\Psi} \log p(\theta_t,\mathbf{h}_t|\mathbf{F},\Psi) \right].
\end{aligned} \tag{4.3}$$

Now, using the fact that $\mathbf{h}_t \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ is independent from $\mathbf{F}$ and Ψ,

$$\begin{aligned}
\nabla_{\mathbf{F},\Psi} \log p(\theta_t,\mathbf{h}_t|\mathbf{F},\Psi) &= \nabla_{\mathbf{F},\Psi} \log \left(p(\theta_t|\mathbf{h}_t,\mathbf{F},\Psi) p(\mathbf{h}_t|\mathbf{F},\Psi) \right) \\
&= \nabla_{\mathbf{F},\Psi} \left(\log p(\theta_t|\mathbf{h}_t,\mathbf{F},\Psi) + \log p(\mathbf{h}_t) \right) \\
&= \nabla_{\mathbf{F},\Psi} \log p(\theta_t|\mathbf{h}_t,\mathbf{F},\Psi).
\end{aligned} \tag{4.4}$$

Substituting Equation (4.4) into Equation (4.3),

$$\nabla_{\mathbf{F},\Psi} \log p(\theta_t|\mathbf{F},\Psi) = \mathbb{E}_{p(\mathbf{h}_t|\theta_t,\mathbf{F},\Psi)} \left[\nabla_{\mathbf{F},\Psi} \log p(\theta_t|\mathbf{h}_t,\mathbf{F},\Psi) \right]. \tag{4.5}$$

Note that $p(\theta_t|\mathbf{h}_t,\mathbf{F},\Psi)$ is just the Gaussian distribution in Equation (2.12), given $\mathbf{F}$ and Ψ. Hence, substituting $\mathbf{c} = \theta_{\text{SWA}}$ into Equation (2.12) and applying the logarithm,

$$\begin{aligned}
\log p(\theta_t|\mathbf{h}_t,\mathbf{F},\Psi) &= -\frac{1}{2}(\theta_t - \mathbf{F}\mathbf{h}_t - \theta_{\text{SWA}})^\intercal \Psi^{-1} (\theta_t - \mathbf{F}\mathbf{h}_t - \theta_{\text{SWA}}) - \frac{1}{2}\log|\Psi| - \frac{D}{2}\log 2\pi \\
&= -\frac{1}{2}(\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)^\intercal \Psi^{-1} (\mathbf{d}_t - \mathbf{F}\mathbf{h}_t) - \frac{1}{2}\log|\Psi| - \frac{D}{2}\log 2\pi,
\end{aligned} \tag{4.6}$$

where $\mathbf{d}_t = \theta_t - \theta_{\text{SWA}}$. This is convenient, since Equation (4.6) can be differentiated with respect to both $\mathbf{F}$ and Ψ. Of course, this requires access to θ_{SWA}, which is not available during training. As a compromise - and following the approach of the SWAG covariance approximation - θ_{SWA} can be replaced by the running average of the neural network's parameter vectors.

4.1.1 Partial derivatives with respect to F

From [23], for any symmetric matrix $\mathbf{W}$,

$$\nabla_{\mathbf{A}} (\mathbf{x} - \mathbf{A}\mathbf{s})^\intercal \mathbf{W} (\mathbf{x} - \mathbf{A}\mathbf{s}) = -2\mathbf{W}(\mathbf{x} - \mathbf{A}\mathbf{s})\mathbf{s}^\intercal. \tag{4.7}$$

Hence, differentiating Equation (4.6) with respect to $\mathbf{F}$ gives

$$\nabla_{\mathbf{F}} \log p(\theta_t|\mathbf{h}_t, \mathbf{F}, \Psi) = \Psi^{-1}(\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)\mathbf{h}_t^{\mathsf{T}}. \tag{4.8}$$

It then follows from Equation (4.5) that $\nabla_{\mathbf{F}} \log p(\theta_t|\mathbf{F}, \Psi)$ is the expected value of $\Psi^{-1}(\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)\mathbf{h}_t^{\mathsf{T}}$ over the distribution $p(\mathbf{h}_t|\theta_t, \mathbf{F}, \Psi)$. Letting $\mathbb{E}[\cdot]$ denote $\mathbb{E}_{p(\mathbf{h}_t|\theta_t,\mathbf{F},\Psi)}[\cdot]$ to simplify the notation,

$$\begin{aligned}\nabla_{\mathbf{F}} \log p(\theta_t|\mathbf{F}, \Psi) &= \mathbb{E}\big[\Psi^{-1}(\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)\mathbf{h}_t^{\mathsf{T}}\big] \\ &= \Psi^{-1}\big(\mathbf{d}_t\mathbb{E}\big[\mathbf{h}_t^{\mathsf{T}}\big] - \mathbf{F}\mathbb{E}\big[\mathbf{h}_t\mathbf{h}_t^{\mathsf{T}}\big]\big).\end{aligned} \tag{4.9}$$

From the E-step of the EM algorithm for FA in [1], $p(\mathbf{h}_t|\theta_t, \mathbf{F}, \Psi) \propto \mathcal{N}(\mathbf{m}_t, \Sigma)$, where

$$\Sigma = (\mathbf{I} + \mathbf{F}^{\mathsf{T}}\Psi^{-1}\mathbf{F})^{-1} \quad \text{and} \quad \mathbf{m}_t = \Sigma\mathbf{F}^{\mathsf{T}}\Psi^{-1}\mathbf{d}_t. \tag{4.10}$$

Hence, using identities for vector and matrix expectations from [23],

$$\mathbb{E}\big[\mathbf{h}_t^{\mathsf{T}}\big] = \mathbf{m}_t^{\mathsf{T}} \quad \text{and} \quad \mathbb{E}\big[\mathbf{h}_t\mathbf{h}_t^{\mathsf{T}}\big] = \Sigma + \mathbf{m}_t\mathbf{m}_t^{\mathsf{T}}. \tag{4.11}$$

Finally, substituting Equation (4.11) into Equation (4.9),

$$\nabla_{\mathbf{F}} \log p(\theta_t|\mathbf{F}, \Psi) = \Psi^{-1}\big(\mathbf{d}_t\mathbf{m}_t^{\mathsf{T}} - \mathbf{F}(\Sigma + \mathbf{m}_t\mathbf{m}_t^{\mathsf{T}})\big). \tag{4.12}$$

4.1.2 Partial derivatives with respect to Ψ

In order to differentiate Equation (4.6) with respect to Ψ, it helps to use the fact that Ψ is diagonal. First consider $\mathbf{X}^{-1} = \operatorname{diag}(\frac{1}{x_1}, \dots, \frac{1}{x_D})$ and $\mathbf{a} = (a_1, \dots, a_D)^{\mathsf{T}}$. Then

$$\mathbf{a}^{\mathsf{T}}\mathbf{X}^{-1}\mathbf{a} = \sum_{d=1}^{D} \frac{a_d^2}{x_d}, \tag{4.13}$$

and so

$$\frac{\partial}{\partial x_d}\mathbf{a}^{\mathsf{T}}\mathbf{X}^{-1}\mathbf{a} = \frac{-a_d^2}{x_d^2} \tag{4.14}$$

for $d = 1, \dots, D$. Since the partial derivatives of Equation (4.13) with respect to the off-diagonal entries of $\mathbf{X}$ are zero,

$$\begin{aligned}\nabla_{\mathbf{X}}(\mathbf{a}^{\mathsf{T}}\mathbf{X}^{-1}\mathbf{a}) &= \operatorname{diag}\Big(\frac{-a_1^2}{x_1^2}, \dots, \frac{-a_D^2}{x_D^2}\Big) \\ &= -\operatorname{diag}\big(\operatorname{diag}(\mathbf{X}^{-2}) \odot \mathbf{a}^2\big),\end{aligned} \tag{4.15}$$

where $\odot$ denotes the element-wise matrix product (with broadcasting, if applicable) and the square of the vector is applied element-wise. Also, when applied to a D-length vector, $\operatorname{diag}(\cdot)$ represents the $D \times D$ diagonal matrix with the vector on its diagonal, and when applied to a $D \times D$ matrix, $\operatorname{diag}(\cdot)$ represents the D-length vector consisting of the diagonal entries of the matrix. Substituting $\mathbf{X} = \Psi$ and $\mathbf{a} = \mathbf{d}_t - \mathbf{F}\mathbf{h}_t$, into Equation (4.15),

$$\nabla_\Psi (\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)^\intercal \Psi^{-1} (\mathbf{d}_t - \mathbf{F}\mathbf{h}_t) = -\operatorname{diag}\left(\operatorname{diag}(\Psi^{-2}) \odot (\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)^2\right). \tag{4.16}$$

Also, using the identity $\nabla_{\mathbf{X}} \log |\mathbf{X}| = \mathbf{X}^{-\intercal}$ from [23] and the fact that $\Psi^{-\intercal} = \Psi^{-1}$,

$$\nabla_\Psi \log |\Psi| = \Psi^{-1}. \tag{4.17}$$

Hence, the partial derivatives of Equation (4.6) with respect to Ψ are

$$\nabla_\Psi \log p(\theta_t | \mathbf{h}_t, \mathbf{F}, \Psi) = \frac{1}{2} \operatorname{diag}\left(\operatorname{diag}(\Psi^{-2}) \odot (\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)^2\right) - \frac{1}{2}\Psi^{-1}. \tag{4.18}$$

Again, letting $\mathbb{E}[\cdot]$ denote $\mathbb{E}_{p(\mathbf{h}_t|\theta_t, \mathbf{F}, \Psi)}[\cdot]$, it follows from Equation (4.5) that

$$\begin{aligned} 2 \cdot \nabla_\Psi \log p(\theta_t | \mathbf{F}, \Psi) &= \mathbb{E}\left[\operatorname{diag}\left(\operatorname{diag}(\Psi^{-2}) \odot (\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)^2\right) - \Psi^{-1}\right] \\ &= \operatorname{diag}\Big(\operatorname{diag}(\Psi^{-2}) \odot \mathbb{E}\left[(\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)^2\right]\Big) - \Psi^{-1}. \end{aligned} \tag{4.19}$$

Now, expanding the expectation inside Equation (4.19) and substituting in the expressions from Equation (4.11),

$$\begin{aligned} \mathbb{E}\left[(\mathbf{d}_t - \mathbf{F}\mathbf{h}_t)^2\right] &= \mathbb{E}\left[\mathbf{d}_t \odot \mathbf{d}_t\right] - 2\mathbb{E}\left[\mathbf{d}_t \odot (\mathbf{F}\mathbf{h}_t)\right] + \mathbb{E}\left[(\mathbf{F}\mathbf{h}_t) \odot (\mathbf{F}\mathbf{h}_t)\right] \\ &= \mathbf{d}_t \odot \mathbf{d}_t - 2\mathbf{d}_t \odot \left(\mathbf{F}\mathbb{E}\left[\mathbf{h}_t\right]\right) + \mathbb{E}\left[\operatorname{diag}(\mathbf{F}\mathbf{h}_t\mathbf{h}_t^\intercal \mathbf{F}^\intercal)\right] \\ &= \mathbf{d}_t \odot \mathbf{d}_t - 2\mathbf{d}_t \odot (\mathbf{F}\mathbf{m}_t) + \operatorname{diag}\left(\mathbf{F}\mathbb{E}\left[\mathbf{h}_t\mathbf{h}_t^\intercal\right]\mathbf{F}^\intercal\right) \\ &= \mathbf{d}_t \odot \mathbf{d}_t - 2\mathbf{d}_t \odot (\mathbf{F}\mathbf{m}_t) + \operatorname{diag}\left(\mathbf{F}(\Sigma + \mathbf{m}_t\mathbf{m}_t^\intercal)\mathbf{F}^\intercal\right). \end{aligned} \tag{4.20}$$

Finally, substituting Equation (4.20) into Equation (4.19) and rearranging,

$$\begin{aligned} \nabla_\Psi \log p(\theta_t | \mathbf{F}, \Psi) = \operatorname{diag}\Bigg(&\frac{1}{2}\operatorname{diag}(\Psi^{-2}) \odot \Big(\mathbf{d}_t \odot \mathbf{d}_t - 2\mathbf{d}_t \odot (\mathbf{F}\mathbf{m}_t) \\ &+ \operatorname{diag}\left(\mathbf{F}(\Sigma + \mathbf{m}_t\mathbf{m}_t^\intercal)\mathbf{F}^\intercal\right)\Big)\Bigg) - \frac{1}{2}\Psi^{-1}. \end{aligned} \tag{4.21}$$

4.1.3 Practical implementation

In practice, storing the full $D \times D$ covariance matrix Ψ (or its inverse) would be infeasible for high-dimensional $\theta_t \in \mathbb{R}^D$. However, since Ψ is diagonal and the partial derivatives of the log-likelihood with respect to the off-diagonal are always zero, it suffices to work with the diagonal entries only. All occurrences of $D \times D$ matrices can then be removed from the gradient calculations. First, note that the partial derivatives with respect to both $\mathbf{F}$ and Ψ depend on $\mathbf{m}_t$ and Σ from Equation (4.10), which themselves depend on Ψ^{-1}. Let $\psi = \operatorname{diag}(\Psi)$ and $\psi^{-n} = \operatorname{diag}(\Psi^{-n})$ for $n \in \mathbb{N}^+$. Then

$$\mathbf{F}^\intercal \Psi^{-1} = (\mathbf{F} \odot \psi^{-1})^\intercal. \tag{4.22}$$

Now, setting $\mathbf{C} = (\mathbf{F} \odot \psi^{-1})^\intercal$ and substituting into Equation (4.10), it follows that

$$\Sigma = (\mathbf{I} + \mathbf{CF})^{-1} \quad \text{and} \quad \mathbf{m}_t = \Sigma \mathbf{C} \mathbf{d}_t. \tag{4.23}$$

These more efficient definitions can then be used in Equation (4.12), which itself can be simplified to

$$\nabla_{\mathbf{F}} \log p(\theta_t | \mathbf{F}, \Psi) = \psi^{-1} \odot \big(\mathbf{d}_t \mathbf{m}_t^\intercal - \mathbf{F}(\Sigma + \mathbf{m}_t \mathbf{m}_t^\intercal)\big). \tag{4.24}$$

Also, the partial derivatives with respect to Ψ in Equation (4.21) can be re-written with respect to ψ, as

$$\begin{aligned} \nabla_\psi \log p(\theta_t | \mathbf{F}, \Psi) = \frac{1}{2} \psi^{-2} \odot \Big(&\mathbf{d}_t \odot \mathbf{d}_t - 2\mathbf{d}_t \odot (\mathbf{F}\mathbf{m}_t) \\ &+ \operatorname{sum}\big((\mathbf{F}(\Sigma + \mathbf{m}_t \mathbf{m}_t^\intercal)) \odot \mathbf{F},\ \dim = 1\big)\Big) - \frac{1}{2}\psi^{-1}, \end{aligned} \tag{4.25}$$

where $\operatorname{sum}(\cdot,\ \dim = 1)$ denotes the operation of summing along the rows of a matrix.

One final point is that the elements of ψ must be positive, since they represent variances. One way to achieve this is by using the re-parameterisation $\psi = \exp \gamma$ for some unconstrained parameter $\gamma \in \mathbb{R}^D$. Then the gradient updates can be performed on γ instead of ψ. Using the chain rule from calculus,

$$\begin{aligned} \nabla_\gamma \log p(\theta_t | \mathbf{F}, \Psi) &= \nabla_\psi \log p(\theta_t | \mathbf{F}, \Psi) \odot \exp \beta \\ &= \nabla_\psi \log p(\theta_t | \mathbf{F}, \Psi) \odot \psi, \end{aligned} \tag{4.26}$$

where $\nabla_\psi \log p(\theta_t | \mathbf{F}, \Psi)$ is given in Equation (4.25). Pseudo code for the practical implementation is given in Algorithm 1.

4.2 Online Expectation-Maximisation

Again, suppose that samples from the posterior, $\theta_1, \theta_2, \ldots, \theta_T$, are available. The classic EM algorithm for FA iteratively optimises the log-likelihood of $\mathbf{F}$ and Ψ by alternating "E" and "M" steps until convergence. Using properties of the Kullback-Leibler

Algorithm 1 Online Stochastic Gradient Ascent for Factor Analysis

```
Input: Observation dimension $D$, latent dimension $K$, learning rate $\alpha > 0$
Initialise $\mathbf{F} \in \mathbb{R}^{D \times K}$, $\psi = \mathbf{1}^D$, $\overline{\theta}_0 = \mathbf{0}^D$
$\gamma \leftarrow \log \psi$
for $t = 1, \ldots, T$ do
    Sample observation $\theta_t \in \mathbb{R}^D$
    $\overline{\theta}_t \leftarrow \overline{\theta}_{t-1} + \frac{1}{t}\big(\overline{\theta}_t - \overline{\theta}_{t-1}\big)$
    $\mathbf{d}_t \leftarrow \theta_t - \overline{\theta}_t$
    $\mathbf{C} \leftarrow (\mathbf{F} \odot \psi^{-1})^\intercal$ (with broadcasting)
    $\Sigma \leftarrow (\mathbf{I} + \mathbf{CF})^{-1}$
    $\mathbf{m}_t \leftarrow \Sigma \mathbf{C} \mathbf{d}_t$
    Compute $\nabla_{\mathbf{F}} \log p(\theta_t | \mathbf{F}, \Psi)$ according to Equation (4.24)
    Compute $\nabla_{\gamma} \log p(\theta_t | \mathbf{F}, \Psi)$ according to Equation (4.26)
    $\mathbf{F} \leftarrow \mathbf{F} + \alpha \nabla_{\mathbf{F}} \log p(\theta_t | \mathbf{F}, \Psi)$
    $\gamma \leftarrow \gamma + \alpha \nabla_{\gamma} \log p(\theta_t | \mathbf{F}, \Psi)$
    $\psi \leftarrow \exp \gamma$
end for
$\theta_{\text{SWA}} \leftarrow \overline{\theta}_T$
return $\theta_{\text{SWA}}, \mathbf{F}, \psi$
```

divergence, it can be shown that

$$L(\mathbf{F},\Psi) \geq -\sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{h}_t|\theta_t)}\left[\log q(\mathbf{h}_t|\theta_t)\right] + \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{h}_t|\theta_t)}\left[\log p(\mathbf{h}_t,\theta_t|\mathbf{F},\Psi)\right], \tag{4.27}$$

where the first and second terms on the right-hand side are called the *entropy* and *energy*, respectively, and $q(\mathbf{h}_t|\theta_t)$, $t = 1,\ldots,T$, are known as the *variational* distributions [1]. The EM algorithm optimises this lower bound on the log-likelihood with respect to $\mathbf{F}$, Ψ and also $q(\mathbf{h}_t|\theta_t)$, hence the name "variational distributions". The idea is that, by pushing up the lower bound, the log-likelihood $L(\mathbf{F},\Psi)$ will hopefully increase as well. In fact, it is guaranteed that each iteration of EM does not decrease $L(\mathbf{F},\Psi)$, which again follows from the properties of the Kullback-Leibler divergence [1].

4.2.1 E-step

In the batch E-step, $\mathbf{F}$ and Ψ are fixed and Equation (4.27) is maximised with respect to $q(\mathbf{h}_t|\theta_t)$, $t = 1,\ldots,T$. From [1], the optimal variational distributions are

$$q(\mathbf{h}_t|\theta_t) = p(\mathbf{h}_t|\theta_t,\mathbf{F},\Psi) \propto \mathcal{N}(\mathbf{m}_t,\Sigma), \tag{4.28}$$

where $\mathbf{m}_t$ and Σ are given in Equation (4.23). Note that this optimisation can be performed separately for each θ_t as it is sampled, using the estimates of $\mathbf{F}$ and Ψ on iteration t. However, in batch EM all $q(\mathbf{h}_t|\theta_t)$ are updated on every iteration. This is clearly not possible in an online algorithm which discards θ_t before sampling θ_{t+1}. Therefore, as a compromise, in the online version each $q(\mathbf{h}_t|\theta_t)$ will be computed once only, on iteration t, and held fixed thereafter. The only other detail is that the batch algorithm uses θ_{SWA} to compute $\mathbf{m}_t$. As θ_{SWA} is not available during training, it will be replaced by the running average of the neural network's parameter vectors, as in Algorithm 1.

4.2.2 M-step

In the batch M-step, $q(\mathbf{h}_t|\theta_t)$, $t = 1,\ldots,T$, are fixed and Equation (4.27) is maximised with respect to $\mathbf{F}$ and Ψ. From [1], the optimal values are $\mathbf{F} = \mathbf{A}\mathbf{H}^{-1}$, where

$$\mathbf{A} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{d}_t\mathbf{m}_t^\mathsf{T} \quad \text{and} \quad \mathbf{H} = \Sigma + \frac{1}{T}\sum_{t=1}^{T} \mathbf{m}_t\mathbf{m}_t^\mathsf{T}, \tag{4.29}$$

and

$$\Psi = \text{diag}\left(\text{diag}\left(\frac{1}{T}\sum_{t=1}^{T} \mathbf{d}_t\mathbf{d}_t^\mathsf{T} - 2\mathbf{F}\mathbf{A}^\mathsf{T} + \mathbf{F}\mathbf{H}\mathbf{F}^\mathsf{T}\right)\right). \tag{4.30}$$

Note that this optimisation involves summing over $t = 1,\dots,T$. Moreover, on each iteration all components of the sums in Equation (4.29) and Equation (4.30) are updated. As in the E-step, updating all components is not possible in an online algorithm. Therefore, in the online version these sums will be updated incrementally on each iteration.

4.2.3 Practical implementation

Let $\overline{\mathbf{A}}_t$ and $\overline{\mathbf{H}}_t$ be the estimates of $\mathbf{A}$ and $\mathbf{H}$ from Equation (4.29), respectively, after iteration t of online EM. That is,

$$\overline{\mathbf{A}}_t = \frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i\mathbf{m}_i^\intercal \quad \text{and} \quad \overline{\mathbf{H}}_t = \Sigma + \frac{1}{t}\sum_{i=1}^{t}\mathbf{m}_i\mathbf{m}_i^\intercal. \tag{4.31}$$

Then the estimates of $\mathbf{F}$ and Ψ on iteration t become $\mathbf{F} = \overline{\mathbf{A}}_t\overline{\mathbf{H}}_t^{-1}$ and

$$\Psi = \operatorname{diag}\left(\operatorname{diag}\left(\frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i\mathbf{d}_i^\intercal - 2\mathbf{F}\overline{\mathbf{A}}_t^\intercal + \mathbf{F}\overline{\mathbf{H}}_t\mathbf{F}^\intercal\right)\right). \tag{4.32}$$

As in Algorithm 1, it suffices to work with $\psi = \operatorname{diag}(\Psi)$ instead of Ψ. The diagonal entries of Equation (4.32) can be computed more efficiently as

$$\begin{aligned}\psi &= \frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i^2 - 2\cdot\operatorname{sum}\big(\mathbf{F}\odot\overline{\mathbf{A}}_t,\ \dim = 1\big) + \operatorname{sum}\big((\mathbf{F}\overline{\mathbf{H}}_t)\odot\mathbf{F},\ \dim = 1\big)\\ &= \frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i^2 + \operatorname{sum}\big((\mathbf{F}\overline{\mathbf{H}}_t)\odot\mathbf{F} - 2\mathbf{F}\odot\overline{\mathbf{A}}_t,\ \dim = 1\big).\end{aligned} \tag{4.33}$$

All that remains to complete the online algorithm is to define the incremental update rules for $\overline{\mathbf{A}}_t$ and $\overline{\mathbf{H}}_t$. Both are similar, and for $\overline{\mathbf{A}}_t$ the derivation is

$$\begin{aligned}\overline{\mathbf{A}}_t &= \frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i\mathbf{m}_i^\intercal\\ &= \frac{1}{t}\left(\frac{t-1}{t-1}\sum_{i=1}^{t-1}\mathbf{d}_i\mathbf{m}_i^\intercal + \mathbf{d}_t\mathbf{m}_t^\intercal\right)\\ &= \frac{1}{t}\left(t\cdot\frac{1}{t-1}\sum_{i=1}^{t-1}\mathbf{d}_i\mathbf{m}_i^\intercal - \frac{1}{t-1}\sum_{i=1}^{t-1}\mathbf{d}_i\mathbf{m}_i^\intercal + \mathbf{d}_t\mathbf{m}_t^\intercal\right)\\ &= \frac{1}{t}\left(t\overline{\mathbf{A}}_{t-1} - \overline{\mathbf{A}}_{t-1} + \mathbf{d}_t\mathbf{m}_t^\intercal\right)\\ &= \overline{\mathbf{A}}_{t-1} + \frac{1}{t}\left(\mathbf{d}_t\mathbf{m}_t^\intercal - \overline{\mathbf{A}}_{t-1}\right).\end{aligned} \tag{4.34}$$

Pseudo code for the practical implementation is given in Algorithm 2.

Algorithm 2 Online Expectation-Maximisation for Factor Analysis

```
Input: Observation dimension $D$, latent dimension $K$
Initialise $\mathbf{F} \in \mathbb{R}^{D\times K}$, $\psi = \mathbf{1}^D$
Initialise $\overline{\theta}_0 = \mathbf{0}^D, \overline{\mathbf{A}}_0 = \mathbf{0}^{D\times K}, \overline{\mathbf{B}}_0 = \mathbf{0}^{K\times K}, \overline{\mathbf{d}^2}_0 = \mathbf{0}^D$
for $t = 1, \dots, T$ do
    Sample observation $\theta_t \in \mathbb{R}^D$
    $\overline{\theta}_t \leftarrow \overline{\theta}_{t-1} + \frac{1}{t}\big(\theta_t - \overline{\theta}_{t-1}\big)$
    $\mathbf{d}_t \leftarrow \theta_t - \overline{\theta}_t$
    $\mathbf{C} \leftarrow (\mathbf{F} \odot \psi^{-1})^\intercal$ (with broadcasting)
    $\Sigma \leftarrow (\mathbf{I} + \mathbf{C}\mathbf{F})^{-1}$
    $\mathbf{m}_t \leftarrow \Sigma\mathbf{C}\mathbf{d}_t$
    $\overline{\mathbf{B}}_t \leftarrow \overline{\mathbf{B}}_{t-1} + \frac{1}{t}(\mathbf{m}_t\mathbf{m}_t^\intercal - \overline{\mathbf{B}}_{t-1})$
    $\overline{\mathbf{H}}_t \leftarrow \Sigma + \overline{\mathbf{B}}_t$
    $\overline{\mathbf{A}}_t \leftarrow \overline{\mathbf{A}}_{t-1} + \frac{1}{t}(\mathbf{d}_t\mathbf{m}_t^\intercal - \overline{\mathbf{A}}_{t-1})$
    $\mathbf{F} \leftarrow \overline{\mathbf{A}}_t\overline{\mathbf{H}}_t^{-1}$
    $\overline{\mathbf{d}^2}_t \leftarrow \overline{\mathbf{d}^2}_{t-1} + \frac{1}{t}(\mathbf{d}_t^2 - \overline{\mathbf{d}^2}_{t-1})$
    $\psi \leftarrow \overline{\mathbf{d}^2}_t + \mathrm{sum}\big((\mathbf{F}\overline{\mathbf{H}}_t) \odot \mathbf{F} - 2\mathbf{F} \odot \overline{\mathbf{A}}_t,\ \mathrm{dim} = 1\big)$
end for
$\theta_{\mathrm{SWA}} \leftarrow \overline{\theta}_T$
return $\theta_{\mathrm{SWA}}, \mathbf{F}, \psi$
```

4.3 Factors Initialisation

One detail missing from Algorithms 1 and 2 is how $\mathbf{F}$ is initialised. To encourage diverse factors, it makes sense to initialise $\mathbf{F}$ with orthogonal columns. In practice, this can be achieved by first generating a matrix $\mathbf{A} \in \mathbb{R}^{D \times K}$, whose entries are independently sampled from a standard normal distribution, and then computing its reduced QR decomposition[1]. This decomposition is $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q} \in \mathbb{R}^{D \times K}$ is orthogonal and $\mathbf{R} \in \mathbb{R}^{K \times K}$ is upper triangular. Then set $\mathbf{F} = \mathbf{Q}$.

4.4 Experiments

The aim of these experiments is to test how well Algorithms 1 and 2 are able to fit observations sampled from actual FA models. The form of a FA model is given in Equation (2.13). In particular, it has parameters $\mathbf{c}, \mathbf{F}$ and Ψ. The maximum likelihood estimate of $\mathbf{c}$, given the data, is just the empirical mean of the sampled observations [1]. This is computed exactly in both algorithms by maintaining a running average of the observations. The other parameters, $\mathbf{F}$ and Ψ, appear in the FA model as part of the full covariance matrix, $\mathbf{F}\mathbf{F}^\intercal + \Psi$. Note that right-multiplying $\mathbf{F}$ by any orthogonal matrix $\mathbf{A} \in \mathbb{R}^{K \times K}$ will result in the exact same covariance [1], since

$$(\mathbf{FA})(\mathbf{FA})^\intercal + \Psi = \mathbf{FAA}^\intercal\mathbf{F}^\intercal + \Psi = \mathbf{FF}^\intercal + \Psi. \tag{4.35}$$

Therefore, the $\mathbf{F}$ estimated by an online FA algorithm cannot be directly compared to the true $\mathbf{F}$. The comparison must be made between the full covariance matrices.

Since the covariance matrices have shape $D \times D$, memory requirements dictate that D, the observation dimension, cannot be too large. Therefore, values of $D = 100$ and $D = 1000$ were used in all experiments in this section. In all cases the latent dimension was set to $K = 10$, meaning that two different observation to latent ratios were tested: $\frac{D}{K} = 10$ and $\frac{D}{K} = 100$. Since computing the maximum likelihood estimate of $\mathbf{c} \in \mathbb{R}^D$ is trivial, in all experiments the entries of $\mathbf{c}$ were simply sampled independently from a standard normal distribution. Each factor loading matrix $\mathbf{F}$ was generated in such a way that its columns spanned the K-dimensional latent space and the conditioning number of the resultant covariance matrix could be controlled. This was achieved by finding the first K eigenvectors of a symmetric positive semi-definite matrix $\mathbf{M} \in \mathbb{R}^{D \times D}$, and then setting the columns of $\mathbf{F}$ equal to these eigenvectors multiplied by a vector $\mathbf{s} > \mathbf{0} \in$

[1] https://pytorch.org/docs/1.9.0/generated/torch.linalg.qr.html

$\mathbb{R}^D$ (element-wise). By construction, the K eigenvectors are linearly independent and therefore span the latent space, and scaling them by $\mathbf{s}$ affects the conditioning number of $\mathbf{F}\mathbf{F}^\intercal$. The conditioning number of a matrix is equal to $\max_{i,j}\{|\lambda_i|/|\lambda_j|\}$, where the λ_i and λ_j are eigenvalues of the matrix [6]. This ratio can be controlled by specifying the range of $\mathbf{s}$, or alternatively, the range of $\mathbf{s}^2$, which is called the *spectrum*. The larger the ratio of the upper to lower bound of the spectrum, the larger the ratio of the eigenvalues. In each experiment the spectrum was sampled from a uniform distribution with one of the following ranges: $[1,10]$, $[1,100]$ or $[1,1000]$. Finally, the diagonal entries of Ψ were sampled from a uniform distribution with upper bound equal to the maximum value of $\mathbf{s}^2$. This is consistent with the FA assumption that an observation, $\theta = \mathbf{F}\mathbf{h} + \mathbf{c} + \varepsilon$, is generated by corrupting the signal $\mathbf{F}\mathbf{h} + \mathbf{c}$ with some random noise $\varepsilon \sim \mathcal{N}(\mathbf{0}, \Psi)$. Pseudo code for generating FA models is given in Algorithm 4 in Appendix B.1.

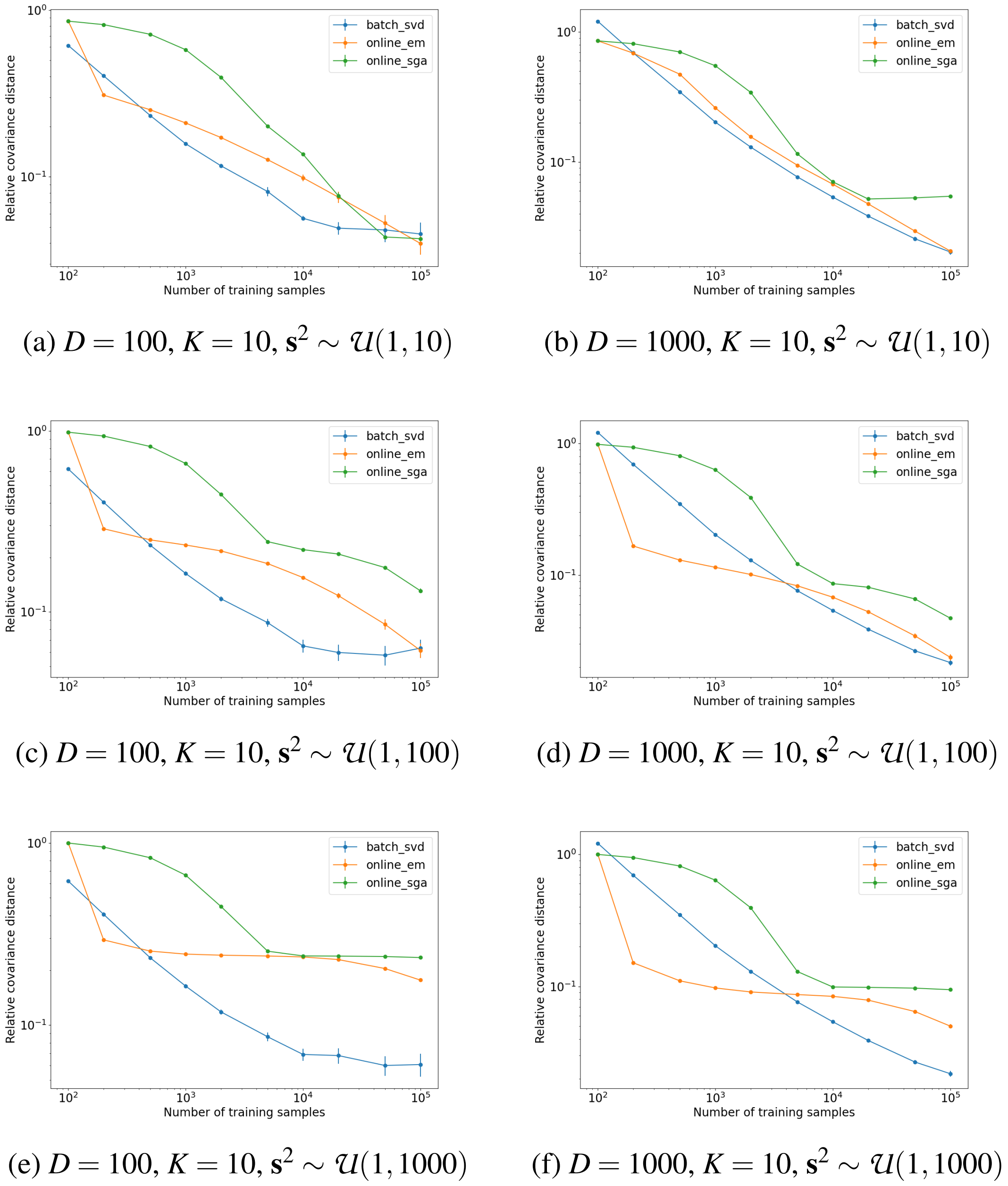
Having generated a FA model, observations were sampled according to Equation (2.11). Using the data, the parameters of the model were then estimated by three separate FA algorithms: online SGA, online EM and the scikit-learn `FactorAnalysis` estimator [22]. The scikit-learn implementation is based on the batch SVD algorithm from [1]. The *randomised* version of SVD was used, which for most applications is "sufficiently precise while providing significant speed gains" [22]. Each algorithm was tested on samples of varying size, from 100 up to 100,000 observations. The latent dimension of each approximate model was set to the true latent dimension K. All other hyperparameters of the scikit-learn algorithm were set to their default values. For both online algorithms, $\mathbf{F}$ was initialised as described in Section 4.3, and online SGA ran with a learning rate of 0.001. Additionally, both online algorithms were allowed a *warm-up* period of 100 iterations during which the running averages, $\overline{\theta}_t, \overline{\mathbf{A}}_t, \overline{\mathbf{B}}_t$ and $\overline{\mathbf{d}^2}_t$, were updated, while $\mathbf{F}$ and Ψ remained fixed. Each experiment was repeated ten times. In each trial a different random seed was used for generating the true FA model and also initialising the parameters of the online algorithms.

4.4.1 Results and Discussion

Figure 4.1 shows the relative distance between the true covariance matrix of each FA model and the corresponding estimated covariance matrix of each learning algorithm, as a function of the number of samples used to fit the approximate models. In the first row of the figure are the results corresponding to FA models whose factor loading matrices were generated with a spectrum range of $[1,10]$. When $D = 100$ (top-left), online EM approximates the true covariance matrix better than online SGA when the number of samples, T, is less than $20,000$. For $T \geq 20,000$, the result are very close and the standard error bars overlap. Moreover, for $T \geq 50,000$, both online algorithms are on par with batch SVD. When $D = 1000$ (top-right), online EM beats SGA for $T \geq 200$ and matches batch SVD when $T = 100,000$. This is in contrast to SGA, which appears to get stuck after $T = 20,000$ and does not decrease the distance any further. In the second row of the figure are the results corresponding to a spectrum range of $[1,100]$. For both $D = 100$ (middle-left) and $D = 1000$ (middle-right), online EM outperforms SGA for $T \geq 200$. Both plots are similar in the sense that online EM initially decreases the distance quite dramatically, even to a greater extent that batch SVD, then the rate of improvement slows before it again catches up with batch SVD at $T = 100,000$. In the third row of the figure are the results corresponding to a spectrum range of $[1,1000]$. In this case, online EM is again superior to SGA. Up until $T = 5000$ the results are similar to those in the second row of the figure. However, in this case online EM does not learn as efficiently as T becomes large and in the end it is not able to match batch SVD. Figure B.1 in Appendix B.1 shows equivalent results for the scaled 2-Wasserstein distance metric.

In summary, these results suggest that online EM is generally better at fitting FA models than online SGA, for almost all combinations of observation dimension, latent dimension, spectrum range and sample size. Compared to batch SVD, the results achieved by online EM are almost identical when $T = 100,000$, except when the spectrum range is equal to $[1,1000]$. This case would appear to be more challenging for online EM due to the larger conditioning number of the covariance matrix. It would be interesting to estimate the conditioning number that could be expected for a deep neural network, but this is beyond the scope of this project.

Given the positive results from these synthetic experiments, online EM would appear a good candidate to learn the posterior distribution of a neural network from its SGD iterates. However, the most that can be expected of online EM is that it fits the SGD iterates well. If the iterates are not representative of the true posterior, applying online EM is a futile endeavour. Unfortunately, it was not possible to obtain representative posterior samples using the method employed by SWAG [17], even for very simple linear regression models (see Appendix B.2 for more details). Therefore, experiments based on SWAG and its variants were curtailed in favour of focusing on a more direct VI approach to posterior estimation, which is described next in Chapter 5.



(a) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1,10)$

(b) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1,10)$

(c) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1,100)$

(d) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1,100)$

(e) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1,1000)$

(f) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1,1000)$

Figure 4.1: The relative distance of the estimated FA covariance matrices from the true covariance matrix as a function of the number of samples used to learn the models. The blue, orange and green lines show, for batch SVD, online EM and online SGA, respectively, the Frobenius norm of the difference between the true covariance matrix and the estimated covariance matrix divided by the Frobenius norm of the true covariance matrix. Each data point shows the mean value over ten trials with different random seeds, and standard error bars are also plotted. The different plots correspond to different combinations of the observation dimension D, the latent dimension K and the range of the spectrum $\mathbf{s}^2$.

Chapter 5

Variational Inference with a Factor Analysis Posterior

This chapter presents VIFA, a VI-based approach to learning a FA posterior of a neural network.

5.1 VIFA

In Section 2.5, a general VI algorithm was outlined for learning an approximate Gaussian posterior of the parameter vector of a neural network. Suppose that the approximate posterior is a FA model. That is, $q(\theta) = \mathcal{N}\big(\mathbf{c}, \mathbf{F}\mathbf{F}^\intercal + \Psi\big)$ for some $\mathbf{c} \in \mathbb{R}^D$, $\mathbf{F} \in \mathbb{R}^{D \times K}$ and $D \times D$ diagonal matrix Ψ. Then the gradient of the VI objective in Equation (2.15) becomes

$$\nabla_{\mathbf{c},\mathbf{F},\Psi}\mathbb{E}_{q(\theta)}[\log q(\theta)] - \nabla_{\mathbf{c},\mathbf{F},\Psi}\mathbb{E}_{q(\theta)}[\log p(\theta)] - \nabla_{\mathbf{c},\mathbf{F},\Psi}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)], \tag{5.1}$$

where the expectation is taken over samples $\theta \sim \mathcal{N}\big(\mathbf{c}, \mathbf{F}\mathbf{F}^\intercal + \Psi\big)$. These derivatives are derived in the following sections.

5.1.1 Partial Derivatives of the Variational Distribution

The logarithm of the pdf of the FA posterior is

$$\log q(\theta) = -\frac{1}{2}(\theta - \mathbf{c})^\intercal(\mathbf{F}\mathbf{F}^\intercal + \Psi)^{-1}(\theta - \mathbf{c}) - \frac{1}{2}\log|\mathbf{F}\mathbf{F}^\intercal + \Psi| - \frac{D}{2}\log 2\pi. \tag{5.2}$$

It is given in [23] that, for any symmetric matrix $\mathbf{A}$ and stochastic vector $\mathbf{x}$ with mean $\mathbf{m}$ and covariance $\mathbf{M}$,

$$\mathbb{E}[\mathbf{x}^\intercal\mathbf{A}\mathbf{x}] = \mathrm{Tr}(\mathbf{A}\mathbf{M}) + \mathbf{m}^\intercal\mathbf{A}\mathbf{m}. \tag{5.3}$$

Setting $\mathbf{S} = \mathbf{F}\mathbf{F}^\mathsf{T} + \Psi$ and noting that $\theta - \mathbf{c}$ has mean $\mathbf{0}$ and covariance $\mathbf{S}$ over $q(\theta)$, it follows that

$$\begin{aligned} \mathbb{E}_{q(\theta)}[\log q(\theta)] &= -\frac{1}{2}\operatorname{Tr}\left(\mathbf{S}^{-1}(\mathbf{F}\mathbf{F}^\mathsf{T} + \Psi)\right) - \frac{1}{2}\mathbf{0}\mathbf{S}^{-1}\mathbf{0} - \frac{1}{2}\log|\mathbf{S}| - \frac{D}{2}\log 2\pi \\ &= -\frac{1}{2}\operatorname{Tr}(\mathbf{I}) - \frac{1}{2}\log|\mathbf{S}| - \frac{D}{2}\log 2\pi \\ &= -\frac{D}{2} - \frac{1}{2}\log|\mathbf{S}| - \frac{D}{2}\log 2\pi. \end{aligned} \tag{5.4}$$

Hence,

$$\nabla_{\mathbf{c}}\mathbb{E}_{q(\theta)}[\log q(\theta)] = \mathbf{0} \tag{5.5}$$

and, using the identity $\nabla_{\mathbf{X}}\log|\mathbf{X}| = \mathbf{X}^{-\mathsf{T}}$ from [23] and the fact that $\mathbf{S}$ is symmetric,

$$\nabla_{\mathbf{S}}\mathbb{E}_{q(\theta)}[\log q(\theta)] = -\frac{1}{2}\mathbf{S}^{-1}. \tag{5.6}$$

It then follows from the chain rule of calculus that

$$\begin{aligned} \nabla_{\mathbf{F}}\mathbb{E}_{q(\theta)}[\log q(\theta)] &= -\frac{1}{2}\mathbf{S}^{-1}(2\mathbf{F}) \\ &= -\mathbf{S}^{-1}\mathbf{F} \end{aligned} \tag{5.7}$$

and

$$\begin{aligned} \nabla_{\Psi}\mathbb{E}_{q(\theta)}[\log q(\theta)] &= -\frac{1}{2}\operatorname{diag}\left(\operatorname{diag}\left(\mathbf{S}^{-1}\mathbf{I}\right)\right) \\ &= -\frac{1}{2}\operatorname{diag}\left(\operatorname{diag}\left(\mathbf{S}^{-1}\right)\right). \end{aligned} \tag{5.8}$$

5.1.2 Partial Derivatives of the Prior

Let the prior be $p(\theta) = \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$ for some precision $\alpha > 0$. Then

$$\begin{aligned} \mathbb{E}_{q(\theta)}[\log p(\theta)] &= \mathbb{E}_{q(\theta)}\left[-\frac{1}{2}\theta^\mathsf{T}\alpha\mathbf{I}\theta - \frac{1}{2}\log|\alpha^{-1}\mathbf{I}| - \frac{D}{2}\log 2\pi\right] \\ &= -\frac{\alpha}{2}\mathbb{E}_{q(\theta)}\left[\theta^\mathsf{T}\mathbf{I}\theta\right] + \text{constant} \\ &= -\frac{\alpha}{2}\left(\operatorname{Tr}(\mathbf{F}\mathbf{F}^\mathsf{T} + \Psi) + \mathbf{c}^\mathsf{T}\mathbf{c}\right) + \text{constant}, \end{aligned} \tag{5.9}$$

where the last line follows from Equation (5.3). Hence,

$$\nabla_{\mathbf{c}}\mathbb{E}_{q(\theta)}[\log p(\theta)] = -\alpha\mathbf{c}, \tag{5.10}$$

$$\nabla_{\mathbf{F}}\mathbb{E}_{q(\theta)}[\log p(\theta)] = -\alpha\mathbf{F}, \tag{5.11}$$

$$\nabla_{\Psi}\mathbb{E}_{q(\theta)}[\log p(\theta)] = -\frac{\alpha}{2}\mathbf{I}, \tag{5.12}$$

which follow from the matrix calculus identities in [23].

5.1.3 Partial Derivatives of the Likelihood

Let the log-likelihood, $\log p(\mathcal{D}|\theta)$, be parameterised by a neural network with weights θ. Due to the functional form of a non-linear neural network, $\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)]$ cannot be computed in closed-form. Instead, a Monte Carlo estimate can be constructed from mini-batches of training data. Formally,

$$\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] \approx \frac{1}{L}\sum_{l=1}^{L}\left(\frac{N}{M}\sum_{(\mathbf{x},y)\in\mathcal{B}_l}\log p(y|\mathbf{x},\theta_l)\right), \tag{5.13}$$

where L is the Monte Carlo average size, each $\mathcal{B}_l$ is a mini-batch of M training examples sampled from the full dataset $\mathcal{D}$ of size N, and $\theta_l \sim q(\theta)$. In order to obtain the partial derivatives of this expression with respect to $\mathbf{c}$, $\mathbf{F}$ and Ψ, the re-parameterisation trick [6] can be used. First note that, by re-writing Equation (2.11), sampling $\theta_l \sim q(\theta) = \mathcal{N}\big(\mathbf{c}, \mathbf{F}\mathbf{F}^\intercal + \Psi\big)$ is equivalent to setting

$$\theta_l = \mathbf{F}\mathbf{h}_l + \mathbf{c} + \Psi^{1/2}\mathbf{z}_l \tag{5.14}$$

from some $\mathbf{h}_l \sim p(\mathbf{h}) = \mathcal{N}(\mathbf{0}^K, \mathbf{I}^{K\times K})$ and $\mathbf{z}_l \sim p(\mathbf{z}) = \mathcal{N}(\mathbf{0}^D, \mathbf{I}^{D\times D})$, where the exponent is applied to Ψ element-wise. It then follows that

$$\begin{aligned}\nabla_{\mathbf{c},\mathbf{F},\Psi}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] &= \nabla_{\mathbf{c},\mathbf{F},\Psi}\mathbb{E}_{p(\mathbf{h})p(\mathbf{z})}[\log p(\mathcal{D}|\theta)] \\ &\approx \frac{1}{L}\sum_{l=1}^{L}\left(N\cdot\nabla_{\mathbf{c},\mathbf{F},\Psi}\left(\frac{1}{M}\sum_{(\mathbf{x},y)\in\mathcal{B}_l}\log p(y|\mathbf{x},\theta_l)\right)\right).\end{aligned} \tag{5.15}$$

Since $\theta_l = \mathbf{F}\mathbf{h}_l + \mathbf{c} + \Psi^{1/2}\mathbf{z}_l$, the chain rule of calculus can be used to obtain these partial derivatives. Formally,

$$\nabla_{\mathbf{c}}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] \approx -\frac{1}{L}\sum_{l=1}^{L}N\mathbf{g}_{\theta_l}, \tag{5.16}$$

$$\nabla_{\mathbf{F}}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] \approx -\frac{1}{L}\sum_{l=1}^{L}N\mathbf{g}_{\theta_l}\mathbf{h}_l^\intercal, \tag{5.17}$$

$$\nabla_{\Psi}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] \approx \operatorname{diag}\left(\operatorname{diag}\left(-\frac{1}{L}\sum_{l=1}^{L}\frac{N}{2}\mathbf{g}_{\theta_l}\mathbf{z}_l^\intercal\Psi^{-1/2}\right)\right), \tag{5.18}$$

where

$$\mathbf{g}_{\theta_l} = \nabla_{\theta_l}\left(-\frac{1}{M}\sum_{(\mathbf{x},y)\in\mathcal{B}_l}\log p(y|\mathbf{x},\theta_l)\right). \tag{5.19}$$

Note that $\mathbf{g}_{\theta_l}$ is the average negative log-likelihood of θ_l for a single training example, where the average is computed over the mini-batch $\mathcal{B}_l$. This is convenient, since the gradient can be computed efficiently using the back-propagation algorithm [26]. Indeed, deep learning frameworks such as PyTorch [21] compute $\mathbf{g}_{\theta_l}$ as standard.

5.1.4 Practical Implementation

In order to compute the gradients in Equations (5.7) and (5.8), the inverse of $\mathbf{S} = \mathbf{F}\mathbf{F}^\intercal + \Psi$ is required. Using the Woodbury identity [23],

$$\begin{aligned}(\mathbf{F}\mathbf{F}^\intercal + \Psi)^{-1} &= \Psi^{-1} - \Psi^{-1}\mathbf{F}(\mathbf{I} + \mathbf{F}^\intercal\Psi^{-1}\mathbf{F})^{-1}\mathbf{F}^\intercal\Psi^{-1} \\ &= \Psi^{-1} - (\psi^{-1} \odot \mathbf{F})\big(\mathbf{I} + \mathbf{F}^\intercal(\psi^{-1} \odot \mathbf{F})\big)^{-1}(\psi^{-1} \odot \mathbf{F})^\intercal \\ &= \Psi^{-1} - \mathbf{A}(\mathbf{I} + \mathbf{F}^\intercal\mathbf{A})^{-1}\mathbf{A}^\intercal,\end{aligned} \tag{5.20}$$

where $\psi = \operatorname{diag}(\Psi)$ and $\mathbf{A} = \psi^{-1} \odot \mathbf{F}$ (with broadcasting). It follows that

$$\begin{aligned}\nabla_{\mathbf{F}}\mathbb{E}_{q(\theta)}[\log q(\theta)] &= -(\mathbf{F}\mathbf{F}^\intercal + \Psi)^{-1}\mathbf{F} \\ &= -\Psi^{-1}\mathbf{F} + \mathbf{A}(\mathbf{I} + \mathbf{F}^\intercal\mathbf{A})^{-1}(\mathbf{A}^\intercal\mathbf{F}) \\ &= -\mathbf{A} + \mathbf{A}(\mathbf{I} + \mathbf{F}^\intercal\mathbf{A})^{-1}(\mathbf{A}^\intercal\mathbf{F}) \\ &= -\mathbf{A} + \mathbf{A}(\mathbf{I} + \mathbf{B})^{-1}\mathbf{B}^\intercal \\ &= -\mathbf{A} + \mathbf{C}\mathbf{B}^\intercal,\end{aligned} \tag{5.21}$$

where $\mathbf{B} = \mathbf{F}^\intercal\mathbf{A}$ and $\mathbf{C} = \mathbf{A}(\mathbf{I} + \mathbf{B})^{-1}$. Also,

$$\begin{aligned}\nabla_{\psi}\mathbb{E}_{q(\theta)}[\log q(\theta)] &= -\frac{1}{2}\operatorname{diag}\left((\mathbf{F}\mathbf{F}^\intercal + \Psi)^{-1}\right) \\ &= -\frac{1}{2}\operatorname{diag}\left(\Psi^{-1} - \mathbf{A}(\mathbf{I} + \mathbf{F}^\intercal\mathbf{A})^{-1}\mathbf{A}^\intercal\right) \\ &= -\frac{1}{2}\psi^{-1} + \frac{1}{2}\operatorname{diag}\left(\mathbf{C}\mathbf{A}^\intercal\right) \\ &= -\frac{1}{2}\psi^{-1} + \frac{1}{2}\operatorname{sum}\big(\mathbf{C} \odot \mathbf{A},\ \dim = 1\big).\end{aligned} \tag{5.22}$$

Note that these calculations involve no explicit use of $D \times D$ matrices. All other references to $D \times D$ matrices can be removed by writing the partial derivatives in Equations (5.12) and (5.18) with respect to ψ. Formally,

$$\nabla_{\psi}\mathbb{E}_{q(\theta)}[\log p(\theta)] = -\frac{\alpha}{2}\mathbf{1}^D, \tag{5.23}$$

$$\nabla_{\psi}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] \approx -\frac{1}{L}\sum_{l=1}^{L}\frac{N}{2}\mathbf{g}_{\theta_l} \odot \left(\psi^{-1/2} \odot \mathbf{z}\right). \tag{5.24}$$

Finally, as in Algorithm 1, the re-parameterisation $\psi = \exp\gamma$ can again be used to ensure that the variances remain positive. Since $\nabla_\gamma\psi = \psi$, it follows from the chain rule that the corresponding partial derivatives with respect to γ are

$$\nabla_{\gamma}\mathbb{E}_{q(\theta)}[\log q(\theta)] = -\frac{1}{2} + \frac{1}{2}\operatorname{sum}\big(\mathbf{C} \odot \mathbf{A},\ \dim = 1\big) \odot \psi, \tag{5.25}$$

$$\nabla_{\gamma}\mathbb{E}_{q(\theta)}[\log p(\theta)] = -\frac{\alpha}{2}\psi, \tag{5.26}$$

$$\nabla_{\gamma}\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] \approx -\frac{1}{L}\sum_{l=1}^{L}\frac{N}{2}\mathbf{g}_{\theta_l} \odot \left(\psi^{1/2} \odot \mathbf{z}\right). \tag{5.27}$$

Collecting the partial derivatives of Equation (5.1), the final update rules are

$$\mathbf{c} \leftarrow \mathbf{c} - \eta_{\mathbf{c}}\left(\alpha\mathbf{c} + \frac{1}{L}\sum_{l=1}^{L} N\mathbf{g}_{\theta_l}\right), \tag{5.28}$$

$$\mathbf{F} \leftarrow \mathbf{F} - \eta_{\mathbf{F}}\left(-\mathbf{A} + \mathbf{C}\mathbf{B}^{\intercal} + \alpha\mathbf{F} + \frac{1}{L}\sum_{l=1}^{L} N\mathbf{g}_{\theta_l}\mathbf{h}_l^{\intercal}\right), \tag{5.29}$$

$$\gamma \leftarrow \gamma - \eta_{\gamma}\left(-\frac{1}{2} + \frac{1}{2}\text{sum}\left(\mathbf{C}\odot\mathbf{A},\ \text{dim}=1\right)\odot\psi + \frac{\alpha}{2}\psi + \frac{1}{L}\sum_{l=1}^{L}\frac{N}{2}\mathbf{g}_{\theta_l}\odot\left(\psi^{1/2}\odot\mathbf{z}\right)\right) \tag{5.30}$$

for learning rates $\eta_{\mathbf{c}}, \eta_{\mathbf{F}}, \eta_{\gamma} > 0$. Pseudo code for the practical implementation is given in Algorithm 3. Being a VI method for approximating a posterior with a FA model, the algorithm is named VIFA.

5.2 Experiments

5.2.1 Posterior Estimation with Synthetic Data

Synthetic data was generated for these experiments as follows. First, 1000 inputs $\mathbf{x} \in \mathbb{R}^2$ were sampled from a multivariate zero mean Gaussian distribution with unit variances and covariances of 0.5. Next, the parameter vector $\theta \in \mathbb{R}^2$ was sampled from $\mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$ with $\alpha = 0.01$. Then the outputs $y \in \mathbb{R}$ were generated according to Equation (2.1) with $\beta = 0.1$. Using this data, the true posterior was evaluated according to Equation (2.6) and an approximate posterior was estimated using VIFA. VIFA ran for 5000 epochs with a batch size of $M = 100$ and a Monte Carlo average size of $L = 10$. The learning rates $\eta_{\mathbf{c}}$, $\eta_{\mathbf{F}}$ and η_{β} were set to 0.01, 0.0001 and 0.01, respectively. The reason for using a smaller learning rate for $\mathbf{F}$ is that its contribution to the full covariance matrix is $\mathbf{F}\mathbf{F}^{\intercal}$. The latent dimension of the approximate FA posterior was set to $K = 1$. When computing the log-likelihood term involved in the gradient calculations, the variance of the outputs y was set to the true value, $\sigma^2 = \frac{1}{\beta} = 10$. Finally, to improve numerical stability, any gradients with Frobenius norm greater than 10 were rescaled to have norm of exactly 10. All experiments were repeated ten times. In each trial a different random seed was used for generating the data, the true parameter vector and the initial parameters of the linear regression model.

Algorithm 3 VIFA

```
Input: Dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, observation dimension $D$, latent dimension $K$,
    prior precision $\alpha > 0$, iterations $T$, mini-batch size $M$, Monte Carlo average size
    $L$, learning rates $\eta_{\mathbf{c}}, \eta_{\mathbf{F}}, \eta_{\gamma} > 0$
 1: Initialise $\mathbf{c} = \mathbf{0}^D$, $\mathbf{F} \in \mathbb{R}^{D \times K}$, $\psi = \mathbf{1}^D$
 2: $\gamma \leftarrow \log \psi$
 3: Initialise $\hat{\mathbf{g}}_{\mathbf{c}} = \mathbf{0}^D$, $\hat{\mathbf{G}}_{\mathbf{F}} = \mathbf{0}^{D \times K}$, $\hat{\mathbf{g}}_{\gamma} = \mathbf{0}^D$
 4: $\mathbf{A} \leftarrow \psi^{-1} \odot \mathbf{F}$ (with broadcasting)
 5: $\mathbf{B} \leftarrow \mathbf{F}^{\intercal}\mathbf{A}$
 6: $\mathbf{C} \leftarrow \mathbf{A}(\mathbf{I} + \mathbf{B})^{-1}$
 7: for $t = 1, \ldots, T$ do
 8:     Sample a mini-batch $\mathcal{B}$ of $M$ training examples from $\mathcal{D}$
 9:     Sample $\mathbf{h} \sim \mathcal{N}(\mathbf{0}^K, \mathbf{I}^{K \times K})$
10:     Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}^D, \mathbf{I}^{D \times D})$
11:     $\theta \leftarrow \mathbf{F}\mathbf{h} + \mathbf{c} + \psi^{1/2} \odot \mathbf{z}$
12:     $\mathbf{g}_{\theta} \leftarrow \nabla_{\theta}\big(-\frac{1}{M}\sum_{(\mathbf{x},y)\in\mathcal{B}} \log p(y|\mathbf{x}, \theta)\big)$ (via back-propagation)
13:     $\hat{\mathbf{g}}_{\mathbf{c}} \leftarrow \hat{\mathbf{g}}_{\mathbf{c}} + N\hat{\mathbf{g}}_{\theta}$
14:     $\hat{\mathbf{G}}_{\mathbf{F}} \leftarrow \hat{\mathbf{G}}_{\mathbf{F}} + N\hat{\mathbf{g}}_{\theta}\mathbf{h}^{\intercal}$
15:     $\hat{\mathbf{g}}_{\gamma} \leftarrow \hat{\mathbf{g}}_{\gamma} + \frac{N}{2}\hat{\mathbf{g}}_{\theta} \odot \big(\psi^{1/2} \odot \mathbf{z}\big)$
16:     if $t \bmod L = 0$ then
17:         $\mathbf{c} \leftarrow \mathbf{c} - \eta_{\mathbf{c}}\big(\alpha\mathbf{c} + \frac{1}{L}\hat{\mathbf{g}}_{\mathbf{c}}\big)$
18:         $\mathbf{F} \leftarrow \mathbf{F} - \eta_{\mathbf{F}}\big(-\mathbf{A} + \mathbf{C}\mathbf{B}^{\intercal} + \alpha\mathbf{F} + \frac{1}{L}\hat{\mathbf{G}}_{\mathbf{F}}\big)$
19:         $\gamma \leftarrow \gamma - \eta_{\gamma}\big(-\frac{1}{2} + \frac{1}{2}\mathrm{sum}(\mathbf{C} \odot \mathbf{A},\ \mathrm{dim} = 1) \odot \psi + \frac{\alpha}{2}\psi + \frac{1}{L}\hat{\mathbf{g}}_{\gamma}\big)$
20:         $\psi \leftarrow \exp\gamma$
21:         Reset $\hat{\mathbf{g}}_{\mathbf{c}}, \hat{\mathbf{G}}_{\mathbf{F}}$ and $\hat{\mathbf{g}}_{\gamma}$ according to line 3
22:         Recalculate $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ according to lines 4-6
23:     end if
24: end for
25: return $\mathbf{c}, \mathbf{F}, \psi$
```

5.2.1.1 Results and Discussion

The results of applying VIFA to the data are shown in Table 5.1. It is interesting to compare these results to those in Appendix B.2, in which the covariance matrix was fit to the SGD iterates. The more direct VI approach does a much better job of approximating the true covariance. The average distance of the estimated covariance from the ground truth is 0.0983 in Table 5.1, which is much lower than the best result of 0.6200 from Table B.3. The improved approximation of the covariance can also be observed in Figure 5.1 for one of the ten VIFA trials.

Relative Distance from Mean	Relative Distance from Covariance	Scaled Wasserstein Distance
0.0031 ± 0.0005	0.0983 ± 0.0129	0.0194 ± 0.0023

Table 5.1: Distances between the true posterior distribution of the parameter vector of a linear regression model and the approximate FA posterior estimated using VIFA. Relative distances between the true and approximate means and covariances are shown. Each relative distance is the Frobenius norm of the difference between the true parameter and the approximate parameter divided by the Frobenius norm of the true parameter. Also shown is the 2-Wasserstein distance between the true posterior Gaussian distribution and the approximate Gaussian distribution, divided by the dimension of the distribution. Each distance is the mean value over ten trials with different random seeds, and standard errors are also given.

5.2.2 Posterior Estimation with UCI Datasets

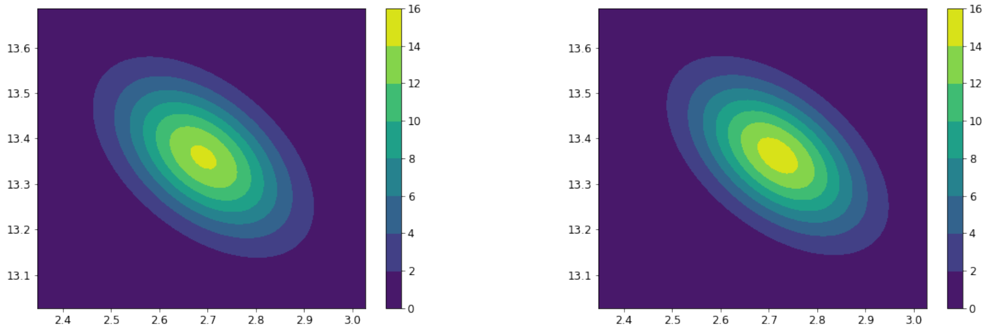
Given the positive results achieved by VIFA in the experiments using synthetic data, the purpose of these experiments is to test how well VIFA is able to approximate the posterior of linear regression models fit to real datasets with more dimensions. For these experiments, four regression datasets from the UCI Machine Learning Repository [5] were used. Namely, the Energy Efficiency[1], Boston Housing[2], Concrete Compressive Strength[3] and Yacht Hydrodynamics[4] datasets. More details about these

[1] https://archive.ics.uci.edu/ml/datasets/energy+efficiency

[2] https://archive.ics.uci.edu/ml/machine-learning-databases/housing

[3] https://archive.ics.uci.edu/ml/datasets/concrete+compressive+strength

[4] http://archive.ics.uci.edu/ml/datasets/yacht+hydrodynamics



(a) Ground truth

(b) VIFA

Figure 5.1: The true posterior pdf of a linear regression model with two learnable parameters, plus the pdf of a FA model with a single latent dimension which was fit to the same data using VIFA.

datasets are given in Table B.4 in Appendix B.3. Each dataset was split into validation and test sets of equal size. The learning rate and number of epochs were tuned on the validation set to minimise the 2-Wasserstein distance between the true and approximate posteriors, and then the algorithm was evaluated once on the test set. To reduce the number of hyperparameters, a single learning rate η was used to optimise all parameters of the approximate posterior. That is, $\eta_{\mathbf{c}} = \eta_{\mathbf{F}} = \eta_{\beta} = \eta$. The latent dimension was set to $K = 3$ for all datasets. Also, the mini-batch size and Monte Carlo average size were set to $M = 100$ and $L = 10$, respectively, since these values worked well for the experiments with synthetic data in Section 5.2.1. The `BayesianRidge` estimator from the scikit-learn library [22] was used to compute the ground truth posterior. This algorithm automatically selects α and β via the Bayesian method described in [16]. These ground truth values were also used when running VIFA. Note, the reason that β is not mentioned explicitly in Algorithm 3 is that no reference was made to the specific form of the likelihood distribution. However, in these linear regression experiments the likelihood is Gaussian and its precision is set to β. Formally, the average negative log-likelihood term used on line 12 of Algorithm 3 was set to

$$-\frac{1}{M} \sum_{(\mathbf{x},y) \in \mathcal{B}} \log \mathcal{N}\big(y; f(\mathbf{x}; \theta), \beta^{-1}\big), \tag{5.31}$$

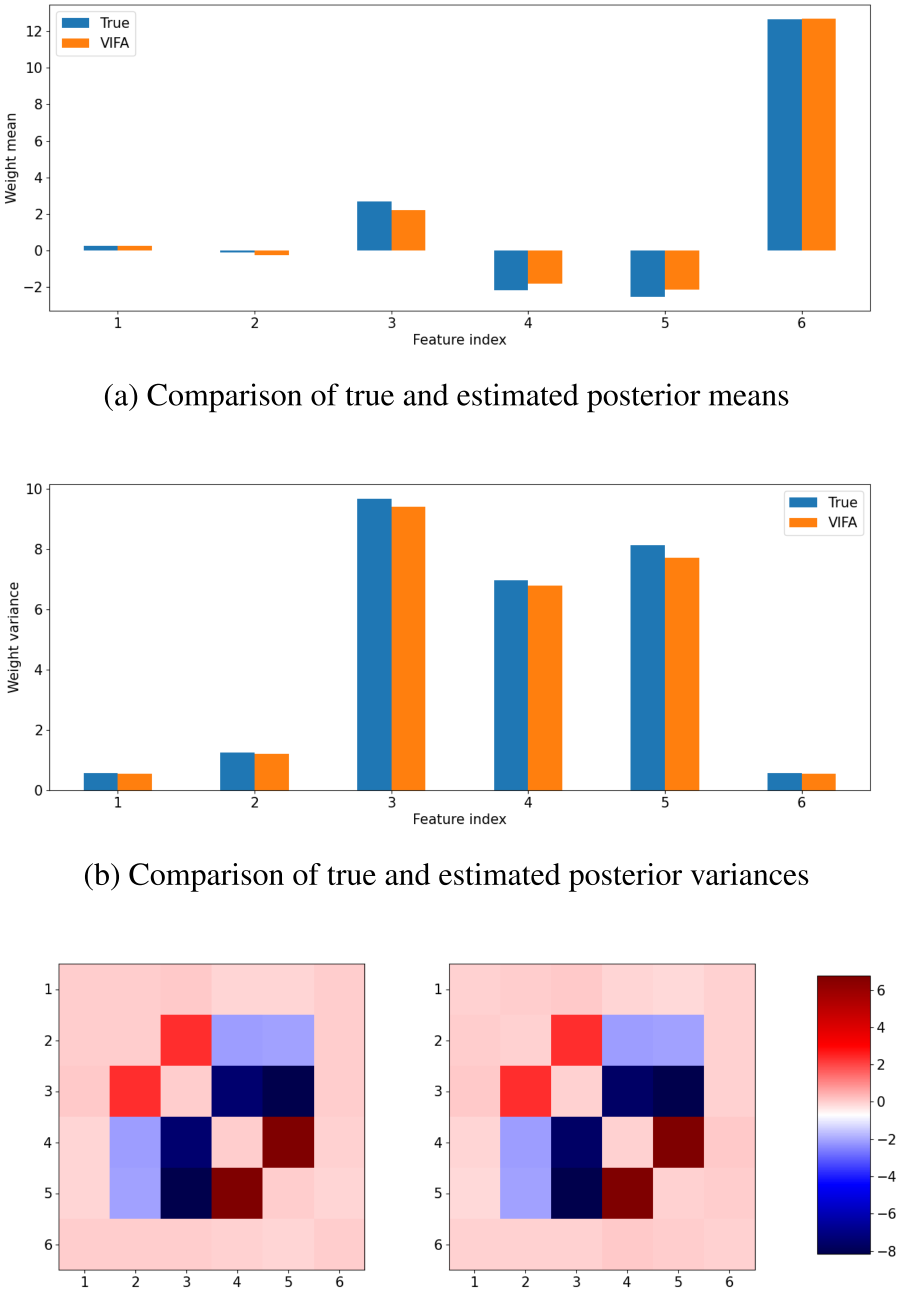
where $\mathcal{B}$ is a mini-batch of M training examples and $f(\mathbf{x}; \theta)$ is the prediction of the linear regression model for input $\mathbf{x}$ and parameters θ. The selected VIFA hyperparameter values for each dataset are given in Table B.5 in Appendix B.3. Finally, before running the algorithms, each input variable was re-scaled to have zero mean and unit standard deviation.

5.2.2.1 Results and Discussion

Figures 5.2 and 5.3 show a qualitative comparison of the true posterior of a linear regression model and the approximate posterior learned by VIFA, in the case of the Yacht Hydrodynamics and Boston Housing datasets, respectively. In the interest of space, qualitative comparisons for the Energy Efficiency and Concrete Compressive Strength datasets can be found in Figures B.4 and B.5 in Appendix B.3, along with quantitative results for all four datasets in Table B.6.

The results for Yacht Hydrodynamics and Boston Housing make for an interesting comparison, since these datasets have dimension 6 and 13, respectively, but in both cases the latent dimension was set to 3. In the case of Yacht Hydrodynamics, Figure 5.2 (c) shows that the off-diagonal entries of the approximate covariance matrix are a good match for the ground truth. In comparison, while the approximate covariance matrix for Boston Housing captures the general structure of the ground truth, the two matrices are noticeably different, as shown in Figure 5.3 (c). On closer inspection, most of the discrepancies appear to be caused by small covariances in the ground truth which are set closer to zero in the approximation. The diagonal entries of the covariance matrix are also better approximated in the case of Yacht Hydrodynamics, as can be seen by comparing Figure 5.2 (b) with Figure 5.3 (b). Although the Boston Housing variances appear to be ordered correctly with respect to each other, most of them are underestimated. Quantitatively, the results in Table B.6 show that the relative distance of the approximate covariance matrix from the ground truth is an order of magnitude less for Yacht Hydrodynamics: 0.0391 compared to 0.3185 for Boston Housing. However, the relative distance of the approximate posterior mean from the ground truth is actually worse for Yacht Hydrodynamics: 0.0435 compared to 0.0252 for Boston Housing. This is not necessarily surprising, since while the quality of the posterior covariance approximation is dependant on the difference between the data dimension and the latent dimension, the posterior mean can in theory be approximated exactly, irrespective of the data dimension.

In summary, across the four datasets, VIFA demonstrates good capacity to approximate the true posterior distribution of linear regressions models. However, the quality of the covariance approximation naturally deteriorates as the data dimension becomes larger in relation to the latent dimension.

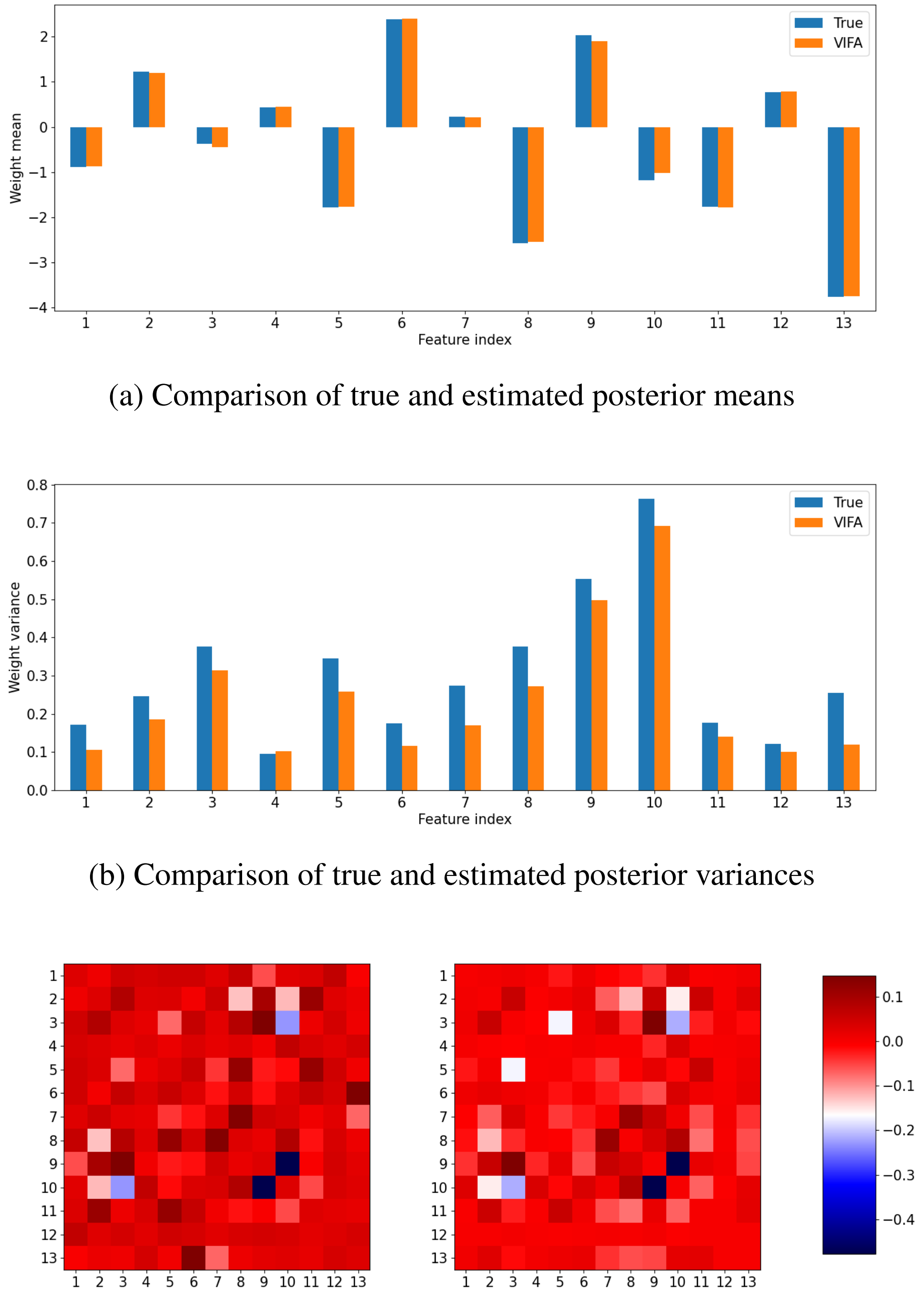


(a) Comparison of true and estimated posterior means

(b) Comparison of true and estimated posterior variances

(c) Comparison of true and estimated posterior covariances

Figure 5.2: Comparison of the ground truth posterior of a linear regression model fit to the Yacht Hydrodynamics dataset, and the approximate posterior learned by VIFA. Variances and covariance are plotted separately due to the difference in their magnitude. In plot (c), the diagonal entries of the covariance matrices have been set to zero.



(a) Comparison of true and estimated posterior means

(b) Comparison of true and estimated posterior variances

(c) Comparison of true and estimated posterior covariances

Figure 5.3: Comparison of the ground truth posterior of a linear regression model fit to the Boston Housing dataset, and the approximate posterior learned by VIFA. Variances and covariance are plotted separately due to the difference in their magnitude. In plot (c), the diagonal entries of the covariance matrices have been set to zero.

5.2.3 Neural Network Predictions

Given the encouraging results achieved by VIFA when applied to linear regression, the purpose of these experiments is to test VIFA on simple neural networks with a single hidden layer. Unlike linear regression, it is not possible to compute the true posterior distribution of the parameter vector of a non-linear neural network in closed form. Therefore, the focus in these experiments is on the quality of predictions made by neural networks trained via VIFA in comparison to SLANG.

The setup for the experiments largely followed the methodology adopted by the authors of SLANG [20]. Each neural network had a single hidden layer with 50 rectified linear units and a single, unactivated output. The UCI regression datasets from Table B.4 were used once again. Each dataset was split 20 times into different train and tests sets[5]. For each train-test split, VIFA hyperparameters were tuned using the training data and then the best configuration was evaluated once on the test set. The hyperparameters which were tuned were the learning rate η, the precision α of the prior and the precision β of the additive target noise distribution. As with SLANG, these were optimised via 30 iterations of hyperparameter search, with 5-fold cross-validation performed in each iteration to estimate the marginal log-likelihood. However, while SLANG used Bayesian optimisation, VIFA was tuned via random search to keep things simple. During the search, values for η, α and β were sampled log-uniformly from $[0.01, 0.02]$, $[0.01, 10]$ and $[0.01, 1]$, respectively. To improve numerical stability, any gradients with Frobenius norm greater than 10 were rescaled to have norm of exactly 10. The other hyperparameters were fixed to the same values as SLANG. That is, latent dimension $K = 1$, number of epochs $T = 120$, mini-batch size $M = 10$ and Monte Carlo average size $L = 4$. Although vanilla SGD is employed in Algorithm 3, a whole host of gradient optimisers can be used in conjunction with VIFA. In this case, the Adam optimiser [13] was used to perform the gradient updates. Before each training run, the features and target were normalised by subtracting their mean and dividing by their standard deviation. These statistics were computed on training data only, including within each separate fold of cross-validation. When training, the average negative Gaussian log-likelihood was used on line 12 of Algorithm 3. When testing, both the average marginal Gaussian log-likelihood and root mean squared error (RMSE) were computed, using a Monte Carlo average size of 100. Full details of how the training loss and test metrics were computed are given in Appendix B.4. The same experiments

[5] Data splits are available at `https://github.com/yaringal/DropoutUncertaintyExps`

were re-run for SLANG[6] with some minor adjustments to the authors' original setup to allow for a fairer comparison with VIFA. In particular, the Monte Carlo average size when testing and the hyperparameter ranges for α and β were adjusted to match those of VIFA. Finally, the VIFA experiments were also run for linear regression models. The only difference being that the hyperparameter range for the learning rate η was set to $[0.001, 0.01]$, as this appeared to work better that $[0.01, 0.02]$ during some initial cross-validation runs.

5.2.3.1 Results and Discussion

Test metrics for VIFA and SLANG, which were aggregated over the 20 train-test splits, are given in Table 5.2. In terms of negative marginal log-likelihood (NMLL), neither algorithm is better than the other on all datasets. For Boston Housing and Concrete Compressive Strength, the mean results for neural networks are essentially the same when standard errors are taken into account. SLANG achieves lower mean NMLL on Yacht Hydrodynamics, but performs extremely poorly on Energy Efficiency with respect to the same metric. It is worth noting that this was caused by a single trial which returned NMLL of 4.04×10^8. This outlier does not affect the median NMLL, which is actually lower than the equivalent value for VIFA. As for mean RMSE, VIFA is the best algorithm on all datasets by some distance. However, SLANG is better on Energy Efficiency and Yacht Hydrodynamics with respect to the median RMSE. The standard errors on the RMSE show that VIFA is very consistent on different train-test splits, whereas the performance of SLANG varies substantially, especially in the case of Energy Efficiency which returned a value of 2.84×10^4 in one trial. One clear advantage that VIFA has over SLANG is speed. In all experiments involving neural networks, the average runtime of VIFA is roughly half that of SLANG. VIFA generally runs slightly faster when learning a linear regression posterior (although, surprisingly, not in the case of Yacht Hydrodynamics). However, the NMLL and RMSE results indicate that there are non-linear relationships in the datasets which are better captured by the neural network.

In summary, the choice between VIFA and SLANG depends on the dataset and evaluation metric. VIFA, though, has the advantage of being more efficient and appears less sensitive to variations in the training and test data.

[6] Using the code available at `https://github.com/aaronpmishkin/SLANG`

Metric	Statistic	Dataset	VIFA-LR	VIFA-NN	SLANG-NN
NMLL	Mean	Energy	**2.57 ± 0.02**	**2.53 ± 0.02**	$2.13 \times 10^7 \pm 2.01 \times 10^7$
		Boston	3.07 ± 0.12	**2.66 ± 0.06**	**2.73 ± 0.06**
		Concrete	3.77 ± 0.02	**3.34 ± 0.01**	**3.32 ± 0.03**
		Yacht	3.65 ± 0.04	2.36 ± 0.06	**1.63 ± 0.04**
	Median	Energy	2.57	2.54	**1.19**
		Boston	2.88	**2.65**	2.68
		Concrete	3.76	3.33	**3.32**
		Yacht	3.62	2.38	**1.59**
RMSE	Mean	Energy	3.07 ± 0.06	**2.90 ± 0.06**	$1.50 \times 10^3 \pm 1.42 \times 10^3$
		Boston	4.64 ± 0.23	**3.64 ± 0.23**	15.87 ± 4.36
		Concrete	10.34 ± 0.15	**6.77 ± 0.09**	22.27 ± 6.64
		Yacht	8.96 ± 0.29	**2.51 ± 0.09**	8.03 ± 4.58
	Median	Energy	3.04	2.93	**1.19**
		Boston	4.31	**3.51**	6.47
		Concrete	10.27	**6.77**	9.03
		Yacht	9.07	2.59	**1.21**
Runtime (minutes)	Mean	Energy	**22.10**	22.59	40.38
		Boston	**14.78**	16.18	32.97
		Concrete	**25.23**	28.41	52.31
		Yacht	11.78	**11.58**	20.61

Table 5.2: Test results for prediction experiments with UCI regression datasets. Metrics are the average negative marginal log-likelihood (NMLL) and root mean squared error (RMSE). LR and NN refer to linear regression and neural network, respectively. The results were aggregated over 20 different train-test splits. The mean results for NMLL and RMSE include standard errors. The runtime refers to the total runtime for a single train-test split. That is, 30 rounds of 5-fold cross-validation on the training set followed by fitting the best model to the full training set and evaluating on the test set. All experiments were executed on an Apple M1 chip (2020) with an 8-core CPU and 16GB of RAM. The best results on each row are highlighted in bold.

Chapter 6

Conclusions

The work in this thesis focused on improving posterior estimation of neural networks. In particular, methods which learn a low-rank plus diagonal approximation of the covariance matrix. Three novel algorithms were presented. Two of these - Algorithms 1 and 2 - can be used to approximate the posterior with a FA model, assuming that representative samples of the posterior can be obtained via some separate procedure. The first method is a stochastic gradient algorithm, whereas the other is based on EM. Crucially, both algorithms are executed online in an iterative fashion and avoid any explicit use of $D \times D$ matrices, ensuring their applicability in the case of very high-dimensional deep neural networks. In the synthetic experiments, the online EM algorithm was shown to be as good as the classic batch SVD algorithm when 10^5 training samples are available and the conditioning number of the true covariance matrix in not too large. Unfortunately, it was not possible to obtain representative samples of the posterior using the approach adopted by SWAG, which curtailed further experiments involving the online FA algorithms. The results in this thesis suggest that SWAG and its variants do not actually do approximate posterior inference and are perhaps better described as ensemble methods. Nevertheless, the online FA algorithms, and in particular online EM, are useful in their own right and could potentially still contribute to Bayesian deep learning if combined with better posterior sampling methods.

Currently, however, the more direct VIFA algorithm appears a much better approach to posterior estimation. This was demonstrated in the experiments involving linear regressions models, fit to both synthetic and real datasets. In all cases, VIFA was able to learn a good approximation of the true posterior covariance matrix. Moreover, VIFA also achieved encouraging results in the neural network predictions experiments in comparison to SLANG, in terms of RMSE, marginal likelihood and runtime.

There are some notable similarities between VIFA and SLANG. Both are VI-based approaches to neural network posterior estimation and both use a low-rank plus diagonal approximation of a $D \times D$ matrix to make the computation tractable - VIFA approximates the posterior covariance whereas SLANG approximates its inverse. Although they are both online gradient-based algorithms, VIFA uses vanilla gradients whereas SLANG uses natural gradients. The authors of SLANG claim that natural gradient methods are preferable when optimising the parameters of a distribution [20], but this leads to undesirable features in practice. In particular, the SLANG update uses a randomised eigen-decomposition algorithm which requires the back-propagated gradients of individual training examples. This is problematic, since an efficient algorithm requires a separate implementation of back-propagation for each different type of neural network layer. Perhaps the biggest advantage of VIFA is that it only requires the gradient of the average negative log-likelihood of a mini-batch of training examples. This is easy to obtain using standard automatic differentiation tools in deep learning frameworks such as PyTorch [21]. Hence, VIFA is model-agnostic and can be applied to any neural network architecture with no extra effort. In this regard, VIFA is a much more practical algorithm that SLANG, as well as being twice as fast.

Given the positive results achieved by VIFA in this thesis, a good next step would be to apply the algorithm to very high-dimensional deep neural networks. VIFA was implemented with this in mind. The fact that Algorithm 3 uses only basic matrix operations and avoids any explicit use of $D \times D$ matrices means that it should be applicable to deep models with millions or even billions of parameters. Running prediction experiments similar to those in Section 5.2.3 but with bigger datasets and bigger models would be an excellent test of the scalability of VIFA. It would also be interesting to include classification datasets in these experiments, since this thesis only dealt with regression. VIFA can be applied in this scenario by simply changing the likelihood distribution from Gaussian to multinomial. Finally, the claim that VIFA is model-agnostic should be put to the test in practice. All experiments in this thesis involved either linear regression models or standard feed-forward neural networks. Further experiments should be executed to test the predictions made by VIFA when applied to other architectures, such as those containing convolutional, recurrent and transformer layers. All these experiments are feasible and would require no modifications to the current implementation of VIFA.

Bibliography

[1] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[2] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. *International conference on machine learning*, pages 1613–1622, 2015.

[3] Scott Brownlie. Informatics project proposal - extending the Bayesian deep learning method MultiSWAG. *University of Edinburgh Informatics Project Proposal*, 2021.

[4] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. CoAtNet: Marrying convolution and attention for all data sizes. *Proceedings of Advances in Neural Information Processing Systems*, 34, 2021.

[5] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[7] Ian J Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[8] Alex Graves. Practical variational inference for neural networks. *Proceeding of Advances in Neural Information Processing Systems*, 24, 2011.

[9] José M Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. *International Conference on Machine Learning*, pages 1861–1869, 2015.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–1780, 1997.

[11] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew G Wilson. Averaging weights leads to wider optima and better generalization. *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*, 2:876–885, 2018.

[12] Mohammad E Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. *Proceedings of the International Conference on Machine Learning*, 35:2611—-2620, 2018.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2012.

[16] David J C MacKay. Bayesian interpolation. *Computation and Neural Systems*, 4:415–447, 1992.

[17] Wesley Maddox, Timur Garipov, Pavel Izmailov, Dmitry Vetrov, and Andrew G Wilson. A simple baseline for bayesian uncertainty in deep learning. *Proceedings of Advances in Neural Information Processing Systems*, 32, 2019.

[18] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate Bayesian inference. *Journal of Machine Learning Research*, 18:4873–4907, 2017.

[19] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[20] Aaron Mishkin, Frederik Kunstner, Didrik Nielsen, Mark Schmidt, and Mohammad E Khan. SLANG: Fast structured covariance approximations for Bayesian deep learning with natural gradient. *Proceedings of Advances in Neural Information Processing Systems*, 31:6248–6258, 2018.

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

Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Proceedings of Advances in Neural Information Processing Systems*, 32:8024–8035, 2019.

[22] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] Kaare B Petersen and Michael S Pedersen. *The Matrix Cookbook*. Technical University of Denmark, 2012.

[24] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. *Artificial Intelligence and Statistics*, pages 814–822, 2014.

[25] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[27] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[28] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Proceedings of Advances in Neural Information Processing Systems*, 30:5998–6008, 2017.

Appendix A

Derivations

A.1 Bayesian Linear Regression Posterior

Applying the logarithm to the prior in Equation (2.5),

$$\begin{aligned}\log p(\theta) &= \frac{D}{2}\log\frac{\alpha}{2\pi} - \frac{\alpha}{2}\theta^{\mathsf{T}}\theta \\ &= -\frac{\alpha}{2}\theta^{\mathsf{T}}\theta + \frac{D}{2}\log\alpha + \text{constant}.\end{aligned} \tag{A.1}$$

Now, using Bayes' rule and Equation (2.4) and Equation (A.1), it follows that the log-posterior distribution of θ for fixed prior precision α and noise precision β is

$$\begin{aligned}\log p(\theta|\mathcal{D}) &= \log\frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\ &= \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D}) \\ &= -\frac{\beta}{2}\sum_{n=1}^{N}\left(y_n - \theta^{\mathsf{T}}\mathbf{x}_n\right)^2 - \frac{\alpha}{2}\theta^{\mathsf{T}}\theta + \text{constant} \\ &= -\frac{\beta}{2}\sum_{n=1}^{N}\left(y_n^2 - 2y_n\theta^{\mathsf{T}}\mathbf{x}_n + (\theta^{\mathsf{T}}\mathbf{x}_n)^2\right) - \frac{\alpha}{2}\theta^{\mathsf{T}}\theta + \text{constant} \\ &= -\frac{\beta}{2}\sum_{n=1}^{N}y_n^2 + \beta\sum_{n=1}^{N}y_n\theta^{\mathsf{T}}\mathbf{x}_n - \frac{\beta}{2}\sum_{n=1}^{N}\theta^{\mathsf{T}}\mathbf{x}_n\mathbf{x}_n^{\mathsf{T}}\theta - \frac{\alpha}{2}\theta^{\mathsf{T}}\theta + \text{constant} \\ &= \left(\beta\sum_{n=1}^{N}y_n\mathbf{x}_n\right)^{\mathsf{T}}\theta - \frac{1}{2}\theta^{\mathsf{T}}\left(\alpha\mathbf{I} + \beta\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\mathsf{T}}\right)\theta + \text{constant} \\ &= \mathbf{b}^{\mathsf{T}}\theta - \frac{1}{2}\theta^{\mathsf{T}}\mathbf{A}\theta + \text{constant},\end{aligned} \tag{A.2}$$

where

$$\mathbf{A} = \alpha\mathbf{I} + \beta\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\mathsf{T}} \quad \text{and} \quad \mathbf{b} = \beta\sum_{n=1}^{N}y_n\mathbf{x}_n. \tag{A.3}$$

Now, using a result from [1] to complete the square of Equation (A.2),

$$\begin{aligned}\log p(\theta|\mathcal{D}) &= -\frac{1}{2}\big(\theta - \mathbf{A}^{-1}\mathbf{b}\big)^{\mathsf{T}}\mathbf{A}\big(\theta - \mathbf{A}^{-1}\mathbf{b}\big) + \frac{1}{2}\mathbf{b}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{b} + \text{constant} \\ &= -\frac{1}{2}\big(\theta - \mathbf{A}^{-1}\mathbf{b}\big)^{\mathsf{T}}\mathbf{A}\big(\theta - \mathbf{A}^{-1}\mathbf{b}\big) + \text{constant} \\ &= -\frac{1}{2}\big(\theta - \mathbf{m}\big)^{\mathsf{T}}\mathbf{S}^{-1}\big(\theta - \mathbf{m}\big) + \text{constant},\end{aligned} \tag{A.4}$$

where

$$\mathbf{m} = \mathbf{A}^{-1}\mathbf{b} \quad \text{and} \quad \mathbf{S} = \mathbf{A}^{-1}. \tag{A.5}$$

Hence, the posterior distribution of θ is

$$p(\theta|\mathcal{D}) = \mathcal{N}(\mathbf{m}, \mathbf{S}). \tag{A.6}$$

Appendix B

Supplementary Results

B.1 Online Factor Analysis

Algorithm 4 shows how the synthetic FA models were generated for the experiments in Section 4.4.

Algorithm 4 Generate a Factor Analysis Model

```
Input: Observation dimension $D$, latent dimension $K$, spectrum range $[a,b]$
Generate $\mathbf{c} \in \mathbb{R}^D$ by sampling entries independently from $\mathcal{N}(0,1)$
Generate $\mathbf{A} \in \mathbb{R}^{D \times D}$ by sampling entries independently from $\mathcal{N}(0,1)$
$\mathbf{M} \leftarrow \mathbf{A}\mathbf{A}^\intercal$
Compute the $K$ eigenvectors, $\mathbf{v}_1, \dots, \mathbf{v}_K \in \mathbb{R}^D$, corresponding to the $K$ largest eigenvalues of $\mathbf{M}$
Construct the matrix $\mathbf{V}_K \in \mathbb{R}^{D \times K}$ with columns $\mathbf{v}_1, \dots, \mathbf{v}_K$
Generate $\mathbf{s}^2 \in \mathbb{R}^D$ by sampling entries independently from $\mathcal{U}(a,b)$
$\mathbf{s} \leftarrow \sqrt{\mathbf{s}^2}$ (square root is applied element-wise)
$\mathbf{F} \leftarrow \mathbf{V}_K \odot \mathbf{s}$ (with broadcasting)
$s_{max} \leftarrow \max(\mathbf{s}^2)$ (largest element of $\mathbf{s}^2$)
Generate $\psi \in \mathbb{R}^D$ by sampling entries independently from $\mathcal{U}(0, s_{max})$
$\Psi \leftarrow \mathrm{diag}(\psi)$
return $\mathbf{c}, \mathbf{F}, \Psi$
```

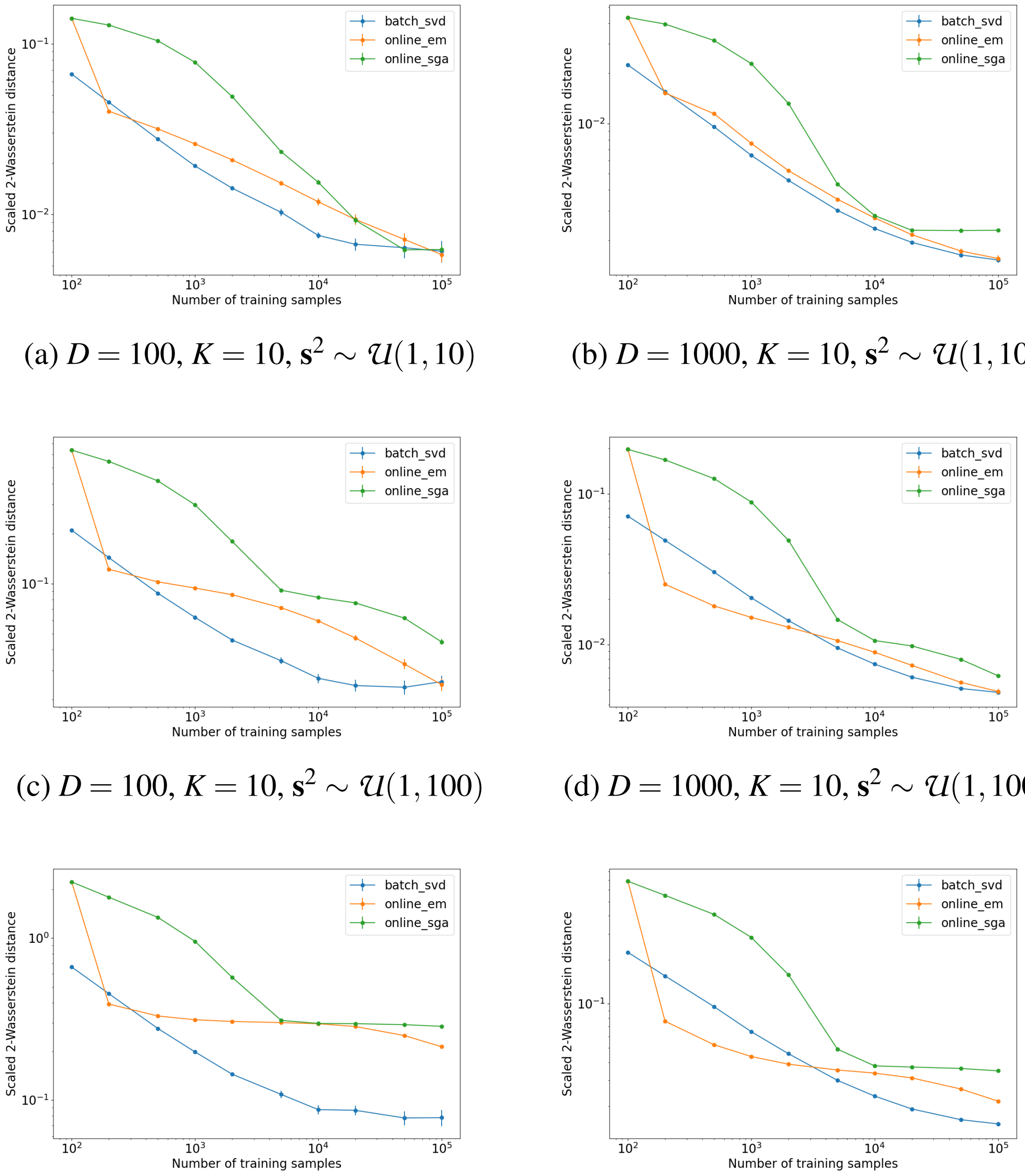
Figure B.1 shows the scaled 2-Wasserstein distance between the Gaussian distribution defined by each estimated FA model and the Gaussian distribution defined by the true FA model, for the same combinations of observation dimension, latent dimension, spectrum range and sample size as Figure 4.1. The distance is scaled by dividing by observation dimension. Tables B.1 and B.2 contain the numerical results corresponding to the data plotted in Figures 4.1 and B.1, respectively, for the case of 100,000 training samples.

Obs. Dim	Spectrum Range	Batch SVD	Online EM	Online SGA
100	$[1,10]$	0.0454 ± 0.0077	0.0399 ± 0.0058	0.0426 ± 0.0005
	$[1,100]$	0.0631 ± 0.0073	0.0613 ± 0.0057	0.1309 ± 0.0038
	$[1,1000]$	0.0608 ± 0.0087	0.1767 ± 0.0037	0.2350 ± 0.0046
1000	$[1,10]$	0.0204 ± 0.0010	0.0207 ± 0.0003	0.0546 ± 0.0009
	$[1,100]$	0.0217 ± 0.0011	0.0238 ± 0.0013	0.0472 ± 0.0008
	$[1,1000]$	0.0218 ± 0.0011	0.0501 ± 0.0016	0.0945 ± 0.0024

Table B.1: The relative distance of the estimated FA covariance matrix from the true covariance matrix when the latent dimension of the FA model is 10 and the number of training samples is 100,000. The relative distance is the Frobenius norm of the difference between the true covariance matrix and the estimated covariance matrix divided by the Frobenius norm of the true covariance matrix. Each result is the mean value over ten trials with different random seeds, along with standard errors. These results correspond to the rightmost points in the plots in Figure 4.1.

B.2 SGD as a Posterior Sampler

The ability of Algorithms 1 and 2 to learn the posterior of a neural network's parameter vector hinges on obtaining samples of the parameter vector which are representative of the true posterior. In SWAG, such samples are thought to be obtained from the iterates of the parameter vector which are encountered along the SGD trajectory while training the neural network with a high constant learning rate, following an initial pre-training phase [17]. In the case of linear regression - which is a neural network with no hidden layers - this assumption can be tested directly, since the true posterior of its parameter vector can be computed in closed form. Recall, however, that the posterior distribution in Equation (2.6) refers to specific values of α and β. The parameter β is $\frac{1}{\sigma^2}$, where σ is the standard deviation of the outputs y_n in Equation (2.3). The precision α is a hyperparameter which controls the width of the prior $p(\theta)$. In Equation (A.2), the



(a) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 10)$

(b) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 10)$

(c) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 100)$

(d) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 100)$

(e) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 1000)$

(f) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 1000)$

Figure B.1: The 2-Wasserstein distance between the Gaussian distribution defined by each estimated FA model and the Gaussian distribution defined by the true FA model, divided by the observation dimension. The blue, orange and green lines show the 2-Wasserstein distance corresponding to batch SVD, online EM and online SGA, respectively. Each data point shows the mean value over ten trials with different random seeds, and standard error bars are also plotted. The different plots correspond to different combinations of the observation dimension D, the latent dimension K and the range of the spectrum $\mathbf{s}^2$.

Obs. Dim	Spectrum Range	Batch SVD	Online EM	Online SGA
100	$[1,10]$	0.0061 ± 0.0009	0.0058 ± 0.0006	0.0062 ± 0.0004
	$[1,100]$	0.0256 ± 0.0022	0.0246 ± 0.0021	0.0446 ± 0.0018
	$[1,1000]$	0.0780 ± 0.0089	0.2137 ± 0.0054	0.2855 ± 0.0040
1000	$[1,10]$	0.0015 ± 0.0001	0.0016 ± 0.0001	0.0023 ± 0.0001
	$[1,100]$	0.0048 ± 0.0002	0.0049 ± 0.0002	0.0062 ± 0.0001
	$[1,1000]$	0.0152 ± 0.0005	0.0217 ± 0.0003	0.0349 ± 0.0006

Table B.2: The 2-Wasserstein distance between the Gaussian distribution defined by the estimated FA model and the Gaussian distribution defined by the true FA model, divided by the observation dimension, when the latent dimension of the FA model is 10 and the number of training samples is 100,000. Each result is the mean value over ten trials with different random seeds, along with standard errors. These results correspond to the rightmost points in the plots in Figure B.1.

log-posterior of θ was written as

$$\log p(\theta|\mathcal{D}) = -\frac{\beta}{2}\sum_{n=1}^{N}\left(y_n - \theta^\mathsf{T}\mathbf{x}_n\right)^2 - \frac{\alpha}{2}\theta^\mathsf{T}\theta + \text{constant}. \tag{B.1}$$

Hence, the maximum *a posteriori* (MAP) estimate of θ is that which maximises

$$-\frac{\beta}{2}\sum_{n=1}^{N}\left(y_n - \theta^\mathsf{T}\mathbf{x}_n\right)^2 - \frac{\alpha}{2}\theta^\mathsf{T}\theta. \tag{B.2}$$

Since $\beta > 0$, it is equivalent to minimise this expression scaled by $-\frac{2}{\beta}$, that is,

$$\sum_{n=1}^{N}\left(y_n - \theta^\mathsf{T}\mathbf{x}_n\right)^2 + \frac{\alpha}{\beta}\theta^\mathsf{T}\theta. \tag{B.3}$$

This objective function is analogous to the L2-regularised training loss often used in non-Bayesian linear regression [1], which is

$$\sum_{n=1}^{N}\left(y_n - \theta^\mathsf{T}\mathbf{x}_n\right)^2 + \lambda\theta^\mathsf{T}\theta \tag{B.4}$$

for some $\lambda > 0$. Setting $\lambda = \frac{\alpha}{\beta}$, the optimal θ found by L2-regularised linear regression is the same as the MAP estimate of θ in Bayesian linear regression with prior $p(\theta) = \mathcal{N}\left(\mathbf{0}, \alpha^{-1}\mathbf{I}\right)$. This means that, given data $\mathcal{D}$ and a particular value of α, the SGD iterates sampled while minimising Equation (B.4) with $\lambda = \frac{\alpha}{\beta}$ should in theory by representative of the true posterior $p(\theta|\mathcal{D})$.

Synthetic data was generated for these experiments in the exact same way as for the experiments in Section 5.2.1. For the purpose of obtaining samples from the posterior, the linear regression model was pre-trained for 500 epochs of SGD with a learning rate of 0.001 and a batch size of 100. Then, starting from the pre-trained model, a further 100 epochs of SGD were executed, during which the parameter vector sampled after each mini-batch gradient update was collected. During the collection period, the batch size was again set to 100 but different values of the learning rate were tested, namely 0.1 and 0.01. As for regularisation, as well as the theoretically motivated value of $\lambda = \frac{\alpha}{\beta} = 0.1$, a value of $\lambda = 0.001$ was also tested. Since there were $\frac{1000}{100} = 10$ batch updates per epoch, a total of 1000 parameter vectors were collected over the final 100 epochs. The empirical mean and covariance of these samples was then computed and compared to the ground truth. All experiments were repeated ten times. In each trial a different random seed was used for generating the data, the true parameter vector and the initial parameters of the linear regression model.

B.2.1 Results and Discussion

The results of the posterior sampling experiments are shown in Table B.3. It is apparent from the third column that the empirical mean of the sampled parameter vectors does not match the true posterior mean when the theoretically motivated value of $\lambda = 0.1$ is used. However, if the regularisation strength is reduced to $\lambda = 0.001$, the distance between the means is much smaller. Unfortunately, the distance between the covariance matrices of the two distributions is relatively large for all combinations of the hyperparameters, as can be seen in the fourth column. The best result is achieved when using a learning rate of 0.1 with $\lambda = 0.1$, but even then the average relative distance is 0.62.

Figure B.2 shows, for each combination of the hyperparameters, the sampled parameter vectors for one of the ten trials in each posterior sampling experiment. These are plotted on top of the contours of the pdf of the true posterior distribution. Note that in each case the pre-trained parameter vector is very close to the true posterior mean. When $\lambda = 0.1$, SGD moves away from the pre-trained solution to a different region of the parameter space (subplots (a) and (c)). When $\lambda = 0.001$, SGD bounces around the pre-trained solution and the empirical mean of the parameter vectors, θ_{SWA}, is right on the true posterior mean. However, when the learning rate is 0.1 the empirical co-

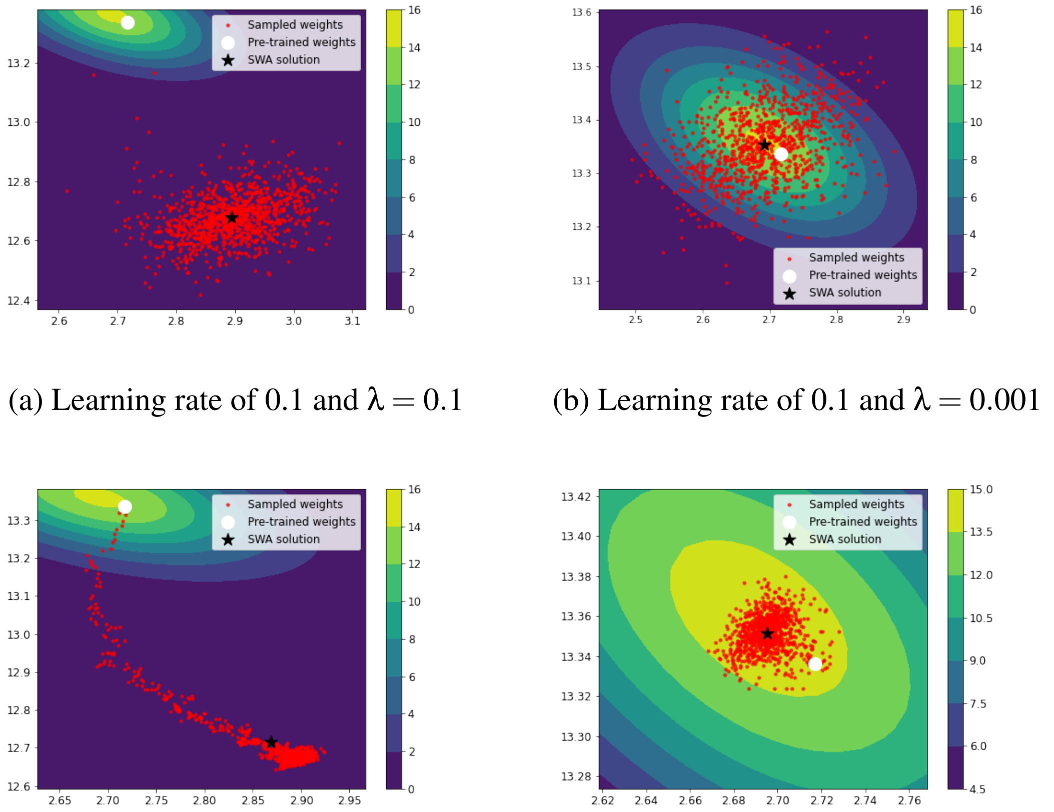
Learning Rate	λ	Relative Distance from Mean	Relative Distance from Covariance	Scaled Wasserstein Distance
0.1	0.1	0.0666 ± 0.0078	0.6200 ± 0.0750	0.4946 ± 0.1013
0.1	0.001	0.0006 ± 0.0001	0.8065 ± 0.0101	0.0439 ± 0.0012
0.01	0.1	0.0620 ± 0.0070	1.7110 ± 0.6187	0.4641 ± 0.0937
0.01	0.001	0.0010 ± 0.0002	0.9870 ± 0.0042	0.0739 ± 0.0010

Table B.3: Distances between the true posterior distribution of the parameter vector of a linear regression model and the empirical distribution of the parameter vectors sampled along the SGD trajectory while training the model. Relative distances between the true and empirical means and covariances are shown. Each relative distance is the Frobenius norm of the difference between the true parameter and the empirical parameter divided by the Frobenius norm of the true parameter. Also shown is the 2-Wasserstein distance between the true posterior Gaussian distribution and the empirical Gaussian distribution, divided by the dimension of the distribution. Each distance is the mean value over ten trials with different random seeds, and standard errors are also given.

variance is in the wrong direction (subplot (b)), and when the learning rate is 0.01 the empirical covariance is far too narrow (subplot (d)).

These results suggest that, in general, SGD does not sample parameter vectors from the true posterior distribution, and hence, SWAG does not actually do approximate Bayesian inference. This is a very simple example of a linear model in two dimensions. Even when tuning the learning rate and regularisation strength, it is not possible to obtain samples which are representative of the true posterior. For certain values of λ (but not the theoretically motivated one), the mean solution θ_{SWA} is a good approximation of the true posterior mean, but the same cannot be said for the empirical covariance. The most that can be expected of the online FA algorithms is that they fit the sampled parameter vectors well. Therefore, given the data observed in Figure B.2, it would be unreasonable to expect them to learn the true posterior distribution. In the case of a high-dimensional multi-layer neural network, approximating the true posterior would be even more unlikely. Since the ground truth cannot be compute in closed form, hyperparameter tuning would be far more difficult.

It was also observed in [18] that the empirical covariance of the linear regression SGD iterates does not match the true posterior covariance. As such, the authors pro-



(a) Learning rate of 0.1 and $\lambda = 0.1$

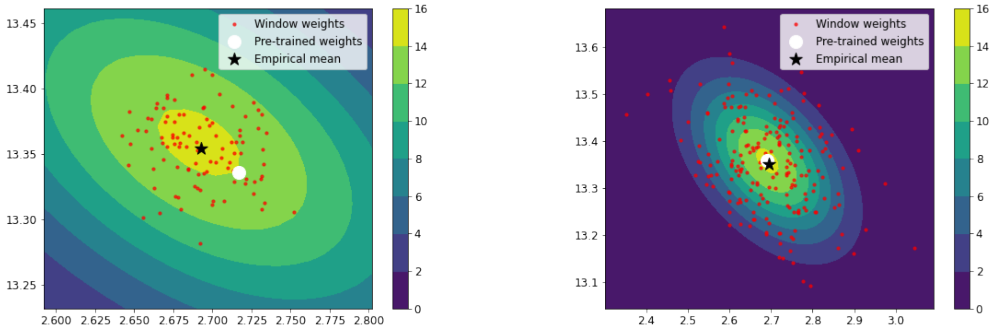
(b) Learning rate of 0.1 and $\lambda = 0.001$

(c) Learning rate of 0.01 and $\lambda = 0.1$

(d) Learning rate of 0.01 and $\lambda = 0.001$

Figure B.2: Parameter vectors sampled while training a linear regression model via SGD, for different combinations of learning rate and regularisation strength. These are shown as red dots, plotted on top of the contours of the pdf of the true posterior distribution. In each plot, the white dot denotes the pre-trained parameter vector and the black star is the empirical mean of the red dots.

posed a different technique for posterior sampling called *finite window iterate averaging*. Instead of using the individual parameter vectors sampled after each mini-batch update of SGD, this algorithm averages the parameter vectors in fixed-length disjoint windows. These average parameter vectors are then considered samples from the posterior. Figure B.3 shows examples of such samples for the same learning rate and regularisation strength as that in Figure B.2 (b). The theoretical work in [18] suggests that the window size should be set to the number of training examples divided by the batch size, so that all training examples are use to generate a single posterior sample. In these experiments this value is $1000/100 = 10$. However, this setting does not produce samples which are representative of the true posterior, as observed in Figure B.3 (a). Better samples can be obtained by setting the batch size to 10 and the window size to 50, as shown in Figure B.3 (b). Unfortunately, this result is not theoretically motivated and was only obtained after significant tuning of both the batch size and window size. In the case of non-linear neural networks for which the true posterior is not known, this would not be possible. Therefore, this algorithm does not seem to be a viable option in practice.



(a) Batch size of 100, window size of 10

(b) Batch size of 10, window size of 50

Figure B.3: Samples constructed via finite window iterate averaging for different combinations of batch size and window size. Each red dot is the mean of a set of linear regression parameter vectors sampled during consecutive mini-batch updates of SGD, with a learning rate of 0.1 and the regularisation strength set to 0.001. The contours show the pdf of the true posterior distribution. The white dot denotes the pre-trained parameter vector and the black star is the empirical mean of the red dots.

B.3 UCI Posterior Estimation

Table B.4 shows the number of instances and and input variables in each UCI regression dataset. Note that the original Energy Efficiency dataset has two target variables, heating load and cooling load, but in these experiments only the heating load was used.

Dataset	No. of Instances	No. of Input Variables
Energy Efficiency	768	8
Boston Housing	506	13
Concrete Compressive Strength	1030	8
Yacht Hydrodynamics	308	6

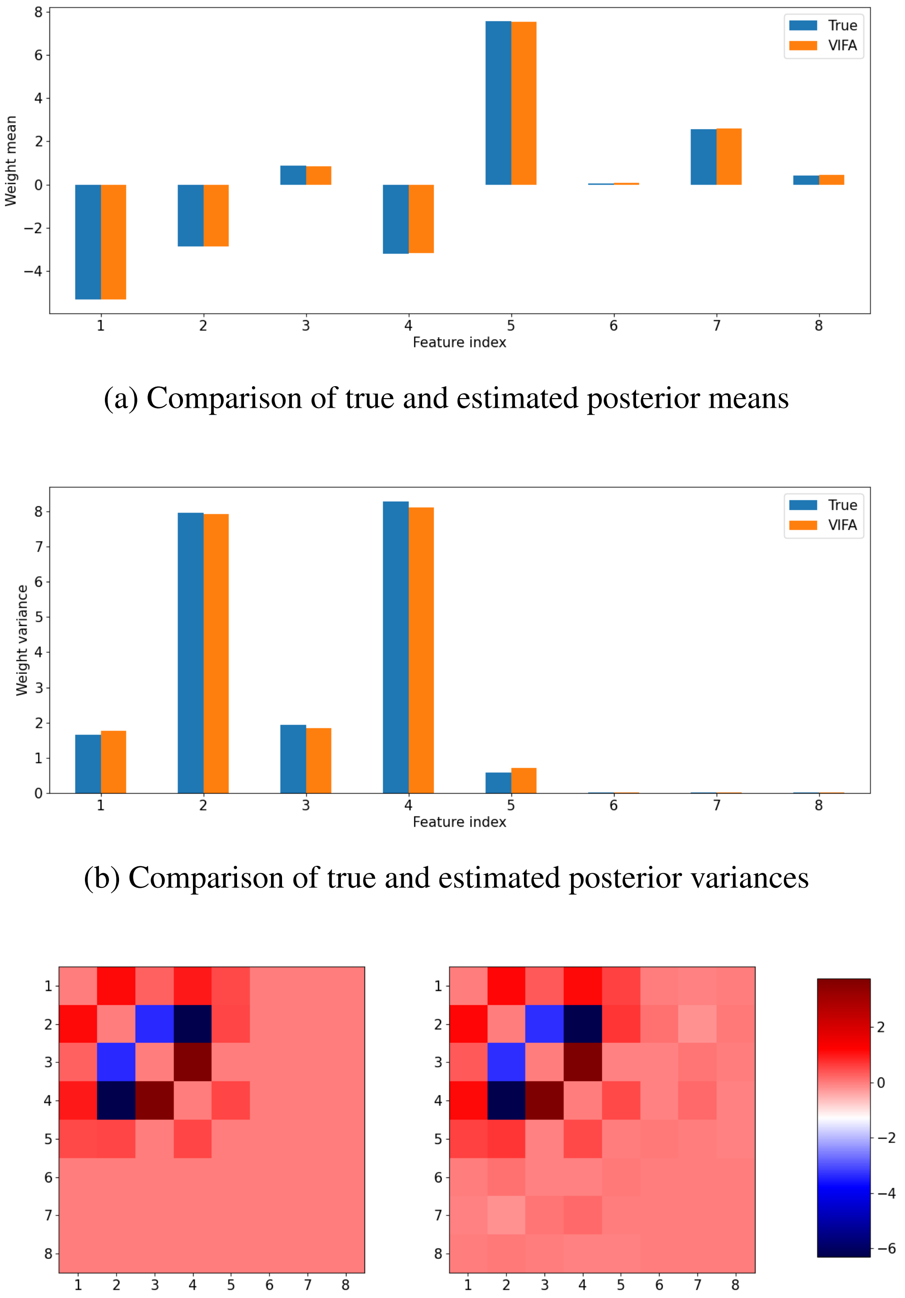
Table B.4: UCI regression datasets used in experiments.

Table B.5 shows the hyperparameter values for the experiments in Section 5.2.2.

Dataset	K	Epochs	M	L	η	α	β
Energy Efficiency	3	25,000	100	10	0.01	0.0608	0.1246
Boston Housing	3	25,000	100	10	0.001	0.2859	0.0429
Concrete Compressive Strength	3	20,000	100	10	0.01	0.0254	0.0101
Yacht Hydrodynamics	3	45,000	100	10	0.01	0.0291	0.0114

Table B.5: VIFA hyperparameters for linear regression posterior estimation experiments with UCI datasets.

Figures B.4 and B.5 show a qualitative comparison of the true posterior of a linear regression model and the approximate posterior learned by VIFA, in the case of the Energy Efficiency and Concrete Compressive Strength datasets, respectively. Table B.6 contains quantitative results for all four UCI datasets.

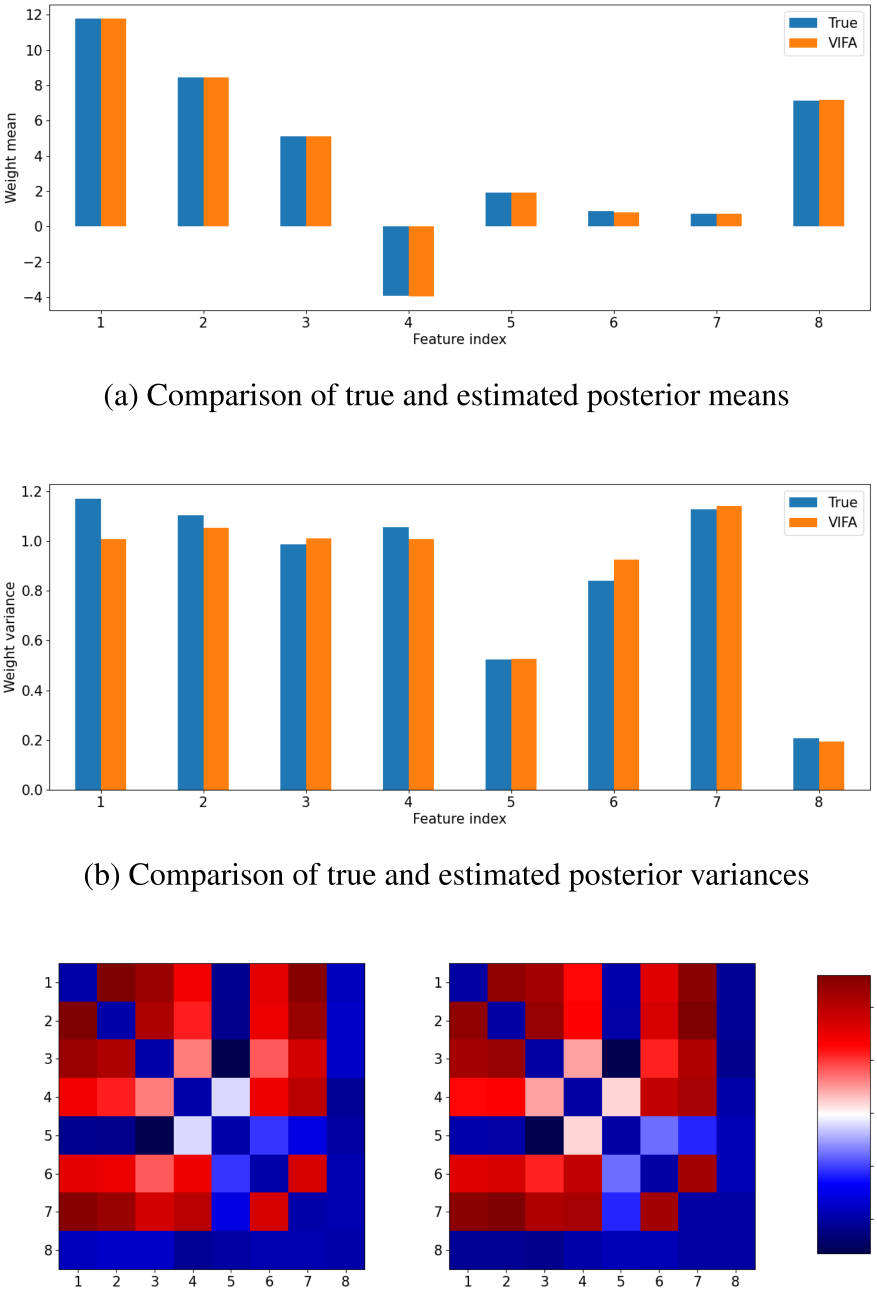


(a) Comparison of true and estimated posterior means

(b) Comparison of true and estimated posterior variances

(c) Comparison of true and estimated posterior covariances

Figure B.4: Comparison of the ground truth posterior of a linear regression model fit to the Energy Efficiency dataset, and the approximate posterior learned by VIFA. Variances and covariance are plotted separately due to the difference in their magnitude. In plot (c), the diagonal entries of the covariance matrices have been set to zero.



(a) Comparison of true and estimated posterior means

(b) Comparison of true and estimated posterior variances

(c) Comparison of true and estimated posterior covariances

Figure B.5: Comparison of the ground truth posterior of a linear regression model fit to the Concrete Compressive Strength dataset, and the approximate posterior learned by VIFA. Variances and covariance are plotted separately due to the difference in their magnitude. In plot (c), the diagonal entries of the covariance matrices have been set to zero.

Dataset	Relative Distance from Mean	Relative Distance from Covariance	Scaled Wasserstein Distance
Energy Efficiency	0.0051	0.0421	0.0564
Boston Housing	0.0262	0.3185	0.0468
Concrete Strength	0.0047	0.0840	0.0278
Yacht Hydro.	0.0435	0.0391	0.1210

Table B.6: For each UCI dataset, the distance between the true posterior distribution of the parameter vector of a linear regression model and the approximate FA posterior estimated by VIFA. Relative distances between the true and approximate means and covariances are shown. Each relative distance is the Frobenius norm of the difference between the true parameter and the approximate parameter divided by the Frobenius norm of the true parameter. Also shown is the 2-Wasserstein distance between the true posterior Gaussian distribution and the approximate Gaussian distribution, divided by the number of dimensions in the dataset.

B.4 UCI Neural Network Predictions

In the experiments in Section 5.2.3, while training, the average negative log-likelihood term used on line 12 of Algorithm 3 was set to

$$-\frac{1}{M}\sum_{(\mathbf{x},y)\in\mathcal{B}} \log \mathcal{N}\big(y; f(\mathbf{x};\theta), \beta^{-1}\big), \tag{B.5}$$

where $\mathcal{B}$ is a mini-batch of M training examples and $f(\mathbf{x};\theta)$ is the prediction of the neural network for input $\mathbf{x}$ and parameters θ. Given a test example, $(\mathbf{x}, y)$, the marginal log-likelihood is

$$\begin{aligned} \log p(y|\mathbf{x},\alpha,\beta,\mathcal{D}) &= \log \int_\theta p(y|\mathbf{x},\theta,\beta)p(\theta|\alpha,\beta,\mathcal{D}) \\ &= \log \int_\theta \mathcal{N}\big(y; f(\mathbf{x};\theta),\beta^{-1}\big)p(\theta|\alpha,\beta,\mathcal{D}), \end{aligned} \tag{B.6}$$

where $\mathcal{D}$ is the training data and $p(\theta|\alpha,\beta,\mathcal{D})$ is the FA posterior learned by VIFA for hyperparameters α and β. This integral cannot be computed exactly. Instead, it was approximated by a sample average. Formally,

$$\begin{aligned}\log p(y|\mathbf{x},\alpha,\beta,\mathcal{D}) &\approx \log\left(\frac{1}{L}\sum_{l=1}^{L}\mathcal{N}\big(y;f(\mathbf{x};\theta_l),\beta^{-1}\big)\right)\\ &= \log\left(\sum_{l=1}^{L}\mathcal{N}\big(y;f(\mathbf{x};\theta_l),\beta^{-1}\big)\right)-\log L,\end{aligned} \tag{B.7}$$

where $\theta_l \sim p(\theta|\alpha,\beta,\mathcal{D})$. This quantity was averaged over the full test set to approximate the average marginal log-likelihood. In addition, the test root mean squared error (RMSE) was also computed, where the squared error of a single test example is

$$\left(y-\frac{1}{L}\sum_{l=1}^{L}f(\mathbf{x};\theta_l)\right)^2. \tag{B.8}$$

When computing these metrics, the Monte Carlo average size was set to $L = 100$.