Extending the Bayesian Deep Learning Method MultiSWAG

Scott Brownlie

Master of Science
School of Informatics
University of Edinburgh
2021

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

Chapter 1

Introduction

The preliminary material of your report should contain:

- The title page.
- An abstract page.
- Optionally an acknowledgements page.
- The table of contents.

As in this example `skeleton.tex`, the above material should be included between:

```
\begin{preliminary}
    ...
\end{preliminary}
```

This style file uses roman numeral page numbers for the preliminary material.

The main content of the dissertation, starting with the first chapter, starts with page 1. ***The main content must not go beyond page 40.***

The report then contains a bibliography and any appendices, which may go beyond page 40. The appendices are only for any supporting material that's important to go on record. However, you cannot assume markers of dissertations will read them.

You may not change the dissertation format (e.g., reduce the font size, change the margins, or reduce the line spacing from the default 1.5 spacing). Over length or incorrectly-formatted dissertations will not be accepted and you would have to modify your dissertation and resubmit. You cannot assume we will check your submission before the final deadline and if it requires resubmission after the deadline to conform to the page and style requirements you will be subject to the usual late penalties based on your final submission time.

Chapter 2

Background

2.1 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

$$\theta = \mathbf{F}\mathbf{h} + \mathbf{c} + \varepsilon. \tag{2.1}$$

In the context of this thesis, an observation θ is the parameter vector of a neural network.

It follows that, given $\mathbf{h}$, the observations θ are Gaussian distributed with mean $\mathbf{F}\mathbf{h} + \mathbf{c}$ and covariance Ψ [1]. Formally,

$$p(\theta|\mathbf{h}) = \mathcal{N}\Big(\mathbf{F}\mathbf{h} + \mathbf{c}, \Psi\Big) = \frac{1}{\sqrt{(2\pi)^D |\Psi|}} \exp\Big(-\frac{1}{2}(\theta - \mathbf{F}\mathbf{h} - \mathbf{c})^\intercal \Psi^{-1} (\theta - \mathbf{F}\mathbf{h} - \mathbf{c})\Big), \tag{2.2}$$

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

$$p(\theta) = \mathcal{N}\big(\mathbf{c}, \mathbf{F}\mathbf{F}^\intercal + \Psi\big). \tag{2.3}$$

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], which in this case is θ_{SWA}. Having set the bias term, an expectation-maximisation (EM)

or singular value decomposition (SVD) algorithm can find the maximum likelihood estimates of $\mathbf{F}$ and Ψ [1]. However, both methods require storing all the observations in memory, making them impractical for high-dimensional data, such as the parameter vectors of deep neural networks. Two alternative online algorithms are presented in Chapter 4.

2.2 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

$$y_n = \theta^\intercal \mathbf{x}_n + \varepsilon \tag{2.4}$$

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

$$\varepsilon \sim \mathcal{N}(0, \sigma^2), \tag{2.5}$$

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 (2.4).

2.2.1 Computing the Posterior

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

$$p(y_n|\theta, \mathbf{x}_n) = \mathcal{N}\big(\theta^\intercal \mathbf{x}_n, \sigma^2\big) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}\big(y_n - \theta^\intercal \mathbf{x}_n\big)^2\Big). \tag{2.6}$$

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

$$\begin{aligned}
\log p(\mathcal{D}|\theta) &= \sum_{n=1}^N \big[\log p(y_n|\theta, \mathbf{x}_n) + \log p(\mathbf{x}_n)\big] \\
&= \sum_{n=1}^N \Big[-\frac{1}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\big(y_n - \theta^\intercal \mathbf{x}_n\big)^2\Big] + \sum_{n=1}^N \log p(\mathbf{x}_n) \\
&= -\frac{1}{2\sigma^2}\sum_{n=1}^N \big(y_n - \theta^\intercal \mathbf{x}_n\big)^2 - \frac{N}{2}\log\sigma^2 - \frac{N}{2}\log 2\pi + \sum_{n=1}^N \log p(\mathbf{x}_n) \\
&= -\frac{\beta}{2}\sum_{n=1}^N \big(y_n - \theta^\intercal \mathbf{x}_n\big)^2 + \frac{N}{2}\log\beta + \text{constant},
\end{aligned} \tag{2.7}$$

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

$$p(\theta) = \mathcal{N}\big(\mathbf{0}, \alpha^{-1}\mathbf{I}\big) = \Big(\frac{\alpha}{2\pi}\Big)^{\frac{D}{2}} \exp\Big(-\frac{\alpha}{2}\theta^\intercal\theta\Big) \tag{2.8}$$

for some $\alpha > 0$, which is a hyperparameter known as the *precision* [1]. Applying the logarithm,

$$\begin{aligned} \log p(\theta) &= \frac{D}{2}\log\frac{\alpha}{2\pi} - \frac{\alpha}{2}\theta^\intercal\theta \\ &= -\frac{\alpha}{2}\theta^\intercal\theta + \frac{D}{2}\log\alpha + \text{constant}. \end{aligned} \tag{2.9}$$

Now, using Bayes' rule and Equation (2.7) and Equation (2.9), it follows that the log-posterior distribution of θ for fixed α and β is

$$\begin{aligned} \log p(\theta|\mathcal{D}) &= \log\frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\ &= \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D}) \\ &= -\frac{\beta}{2}\sum_{n=1}^{N}\big(y_n - \theta^\intercal\mathbf{x}_n\big)^2 - \frac{\alpha}{2}\theta^\intercal\theta + \text{constant} \\ &= -\frac{\beta}{2}\sum_{n=1}^{N}\big(y_n^2 - 2y_n\theta^\intercal\mathbf{x}_n + (\theta^\intercal\mathbf{x}_n)^2\big) - \frac{\alpha}{2}\theta^\intercal\theta + \text{constant} \\ &= -\frac{\beta}{2}\sum_{n=1}^{N}y_n^2 + \beta\sum_{n=1}^{N}y_n\theta^\intercal\mathbf{x}_n - \frac{\beta}{2}\sum_{n=1}^{N}\theta^\intercal\mathbf{x}_n\mathbf{x}_n^\intercal\theta - \frac{\alpha}{2}\theta^\intercal\theta + \text{constant} \\ &= \Big(\beta\sum_{n=1}^{N}y_n\mathbf{x}_n\Big)^\intercal\theta - \frac{1}{2}\theta^\intercal\Big(\alpha\mathbf{I} + \beta\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^\intercal\Big)\theta + \text{constant} \\ &= \mathbf{b}^\intercal\theta - \frac{1}{2}\theta^\intercal\mathbf{A}\theta + \text{constant}, \end{aligned} \tag{2.10}$$

where

$$\mathbf{A} = \alpha\mathbf{I} + \beta\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^\intercal \quad \text{and} \quad \mathbf{b} = \beta\sum_{n=1}^{N}y_n\mathbf{x}_n. \tag{2.11}$$

Now, using a result from [1] to complete the square of Equation (2.10),

$$\begin{aligned} \log p(\theta|\mathcal{D}) &= -\frac{1}{2}\big(\theta - \mathbf{A}^{-1}\mathbf{b}\big)^\intercal\mathbf{A}\big(\theta - \mathbf{A}^{-1}\mathbf{b}\big) + \frac{1}{2}\mathbf{b}^\intercal\mathbf{A}^{-1}\mathbf{b} + \text{constant} \\ &= -\frac{1}{2}\big(\theta - \mathbf{A}^{-1}\mathbf{b}\big)^\intercal\mathbf{A}\big(\theta - \mathbf{A}^{-1}\mathbf{b}\big) + \text{constant} \\ &= -\frac{1}{2}\big(\theta - \mathbf{m}\big)^\intercal\mathbf{S}^{-1}\big(\theta - \mathbf{m}\big) + \text{constant}, \end{aligned} \tag{2.12}$$

where

$$\mathbf{m} = \mathbf{A}^{-1}\mathbf{b} \quad \text{and} \quad \mathbf{S} = \mathbf{A}^{-1}. \tag{2.13}$$

Hence, the posterior distribution of θ is

$$p(\theta|\mathcal{D}) = \mathcal{N}(\mathbf{m}, \mathbf{S}). \tag{2.14}$$

2.3 Variational Inference

For linear regression, computing the posterior in Equation (2.14) 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 directly with a Gaussian distribution $q(\theta) = \mathcal{N}(\mu, \Sigma)$, where the parameters μ and Σ are chosen to minimise some measure of distance 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 distance between two distributions is the Kullback-Leibler (KL) divergence [1], which in this case is

$$\begin{aligned}\mathbb{D}_{KL}[q(\theta)\|p(\theta|\mathcal{D})] &= \mathbb{E}_{q(\theta)}\Big[\log \frac{q(\theta)}{p(\theta|\mathcal{D})}\Big] \\ &= \mathbb{E}_{q(\theta)}\Big[\log \frac{q(\theta)p(\mathcal{D})}{p(\mathcal{D}|\theta)p(\theta)}\Big] \\ &= \mathbb{E}_{q(\theta)}[\log q(\theta) - \log p(\theta) - \log p(\mathcal{D}|\theta) + \log p(\mathcal{D})].\end{aligned} \tag{2.15}$$

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

$$\min_{\mu,\Sigma} \big[\mathbb{E}_{q(\theta)}[\log q(\theta)] - \mathbb{E}_{q(\theta)}[\log p(\theta)] - \mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)]\big]. \tag{2.16}$$

One way to do this is by using a stochastic gradient algorithm, which approximates the true gradient of the objective function with Monte Carlo estimates of the form

$$\frac{1}{ML}\sum_{m=1}^{M}\sum_{l=1}^{L}\big[\nabla_{\mu,\Sigma}\log q(\theta_l) - \nabla_{\mu,\Sigma}\log p(\theta_l) - \nabla_{\mu,\Sigma}\log p(y_m|\mathbf{x}_m,\theta_l)\big], \tag{2.17}$$

where $\{(y_m, \mathbf{x}_m)\}_{m=1}^M$ is a batch of observed data sampled from $\mathcal{D}$ and $\theta_l \sim \mathcal{N}(\mu, \Sigma)$. Normally it would not be possible to compute partial derivatives with respect to μ and Σ, given that they only take part via the random sampling operation. However, this can in fact be done by using the *re-parameterisation trick* [3]. Instead of sampling 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 (2.17) can be evaluated and used in conjunction with SGD or any of its variants to optimise the objective in Equation (2.16). This is the idea behind the *auto-encoding variational Bayes* algorithm [4].

Chapter 3

Related Work and Research Questions

Chapter 4

Online Factor Analysis

This chapter describes three algorithms which can be used to approximate the posterior distribution of the parameter vector of a neural network with a FA model. The first two algorithms fit a FA model to samples of the posterior obtained via some separate procedure, while the third algorithm estimates the FA posterior directly. Crucially, all three algorithms are executed online in an iterative fashion, making them suitable for high-dimensional deep neural networks.

4.1 Online Stochastic Gradient Ascent

Suppose first that samples from the posterior, $\theta_1, \ldots, \theta_T$, 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, $\theta_1, \ldots, \theta_T$. 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 $\nabla_{\mathbf{F},\Psi} \log p(\theta_t | \mathbf{F}, \Psi)$, $t = 1, \ldots, T$, 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.

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

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

This is as far as the derivation in [1] goes. However, given the form of the FA model, it is possible to manipulate the sample derivatives further. In particular, using the fact that $\mathbf{h}_t \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ is independent from $\mathbf{F}$ and Ψ,

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

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 (2.2), given $\mathbf{F}$ and Ψ. Hence, substituting $\mathbf{c} = \theta_{\text{SWA}}$ into Equation (2.2) 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 $\mathbf{F}$

From [8], 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

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

It then follows from Equation (4.5) that $\nabla_{\mathbf{F}} \log p(\theta_t|\mathbf{F},\Psi)$ is the expected value of $\Psi^{-1}(\mathbf{d}_t-\mathbf{F}\mathbf{h}_t)\mathbf{h}_t^{\intercal}$ 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,

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

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

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

Hence, using identities from [8],

$$\mathbb{E}\left[\mathbf{h}_t^{\intercal}\right] = \mathbf{m}_t^{\intercal} \quad \text{and} \quad \mathbb{E}\left[\mathbf{h}_t\mathbf{h}_t^{\intercal}\right] = \Sigma+\mathbf{m}_t\mathbf{m}_t^{\intercal}. \tag{4.11}$$

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

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

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 $\mathbf{X}^{-1} = \text{diag}(\frac{1}{x_1},\ldots,\frac{1}{x_D})$ and $\mathbf{a} = (a_1,\ldots,a_D)^{\intercal}$. Then

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

and so

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

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

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

where $\odot$ denotes the element-wise matrix product (with broadcasting, if applicable) and the square of a 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),

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

Also, using the identity $\nabla_{\mathbf{X}}\log|\mathbf{X}| = \mathbf{X}^{-\intercal}$ from [8] 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

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

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

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

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

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

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

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

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

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

These more efficient values 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 $\psi = \exp\beta$ for

some unconstrained parameter $\beta \in \mathbb{R}^D$. Then the gradient updates can be performed on β instead of ψ. Using the chain rule from calculus,

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

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.

Algorithm 1 Online Stochastic Gradient Ascent for Factor Analysis

Input: Observation dimension D, latent dimension K, learning rate $\alpha > 0$

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

4.2 Online Expectation-Maximisation

Again suppose that samples from the posterior, $\theta_1,\ldots,\theta_T$, 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

divergence, it can be shown that

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

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 the online SGA algorithm from Section 4.1.

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

$$\mathbf{F} = \mathbf{A}\mathbf{H}^{-1}, \tag{4.29}$$

where

$$\mathbf{A} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{d}_t\mathbf{m}_t^\intercal \quad \text{and} \quad \mathbf{H} = \Sigma + \frac{1}{T}\sum_{t=1}^{T} \mathbf{m}_t\mathbf{m}_t^\intercal, \tag{4.30}$$

and

$$\Psi = \operatorname{diag}\left(\operatorname{diag}\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{d}_t\mathbf{d}_t^\intercal - 2\mathbf{F}\mathbf{A}^\intercal + \mathbf{F}\mathbf{H}\mathbf{F}^\intercal\right)\right). \tag{4.31}$$

Note that this optimisation involves summing over $t = 1, \ldots, T$. Moreover, on each iteration all components of the sums in Equation (4.30) and Equation (4.31) 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 (4.30), respectively, after iteration t of online EM. That is,

$$\overline{\mathbf{A}}_t = \frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i\mathbf{m}_i^\intercal \quad \text{and} \quad \overline{\mathbf{H}}_t = \Sigma + \frac{1}{t}\sum_{i=1}^{t}\mathbf{m}_i\mathbf{m}_i^\intercal. \tag{4.32}$$

Then the estimates of $\mathbf{F}$ and Ψ on iteration t become

$$\mathbf{F} = \overline{\mathbf{A}}_t\overline{\mathbf{H}}_t^{-1} \tag{4.33}$$

and

$$\Psi = \operatorname{diag}\left(\operatorname{diag}\left(\frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i\mathbf{d}_i^\intercal - 2\mathbf{F}\overline{\mathbf{A}}_t^\intercal + \mathbf{F}\overline{\mathbf{H}}_t\mathbf{F}^\intercal\right)\right). \tag{4.34}$$

As in Algorithm 1, it suffices to work with $\psi = \operatorname{diag}(\Psi)$ instead of Ψ. The diagonal entries of Equation (4.34) can be computed more efficiently as

$$\begin{aligned}\psi &= \frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i^2 - 2\cdot\operatorname{sum}\big(\mathbf{F}\odot\overline{\mathbf{A}}_t,\ \dim = 1\big) + \operatorname{sum}\big((\mathbf{F}\overline{\mathbf{H}}_t)\odot\mathbf{F},\ \dim = 1\big)\\ &= \frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i^2 + \operatorname{sum}\big((\mathbf{F}\overline{\mathbf{H}}_t)\odot\mathbf{F} - 2\mathbf{F}\odot\overline{\mathbf{A}}_t,\ \dim = 1\big).\end{aligned} \tag{4.35}$$

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

$$\begin{aligned}\overline{\mathbf{A}}_t &= \frac{1}{t}\sum_{i=1}^{t}\mathbf{d}_i\mathbf{m}_i^\intercal\\ &= \frac{1}{t}\left(\frac{t-1}{t-1}\sum_{i=1}^{t-1}\mathbf{d}_i\mathbf{m}_i^\intercal + \mathbf{d}_t\mathbf{m}_t^\intercal\right)\\ &= \frac{1}{t}\left(t\cdot\frac{1}{t-1}\sum_{i=1}^{t-1}\mathbf{d}_i\mathbf{m}_i^\intercal - \frac{1}{t-1}\sum_{i=1}^{t-1}\mathbf{d}_i\mathbf{m}_i^\intercal + \mathbf{d}_t\mathbf{m}_t^\intercal\right)\\ &= \frac{1}{t}\left(t\overline{\mathbf{A}}_{t-1} - \overline{\mathbf{A}}_{t-1} + \mathbf{d}_t\mathbf{m}_t^\intercal\right)\\ &= \overline{\mathbf{A}}_{t-1} + \frac{1}{t}\left(\mathbf{d}_t\mathbf{m}_t^\intercal - \overline{\mathbf{A}}_{t-1}\right).\end{aligned} \tag{4.36}$$

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 $t = 1, \ldots, T$ **do**
    Sample observation $\theta_t \in \mathbb{R}^D$
    $\overline{\theta}_t \leftarrow \overline{\theta}_{t-1} + \frac{1}{t}\left(\overline{\theta}_t - \overline{\theta}_{t-1}\right)$
    $\mathbf{d}_t \leftarrow \theta_t - \overline{\theta}_t$
    $\mathbf{C} \leftarrow (\mathbf{F} \odot \psi^{-1})^\intercal$ (with broadcasting)
    $\Sigma \leftarrow (\mathbf{I} + \mathbf{C}\mathbf{F})^{-1}$
    $\mathbf{m}_t \leftarrow \Sigma \mathbf{C} \mathbf{d}_t$
    $\overline{\mathbf{B}}_t \leftarrow \overline{\mathbf{B}}_{t-1} + \frac{1}{t}(\mathbf{m}_t \mathbf{m}_t^\intercal - \overline{\mathbf{B}}_{t-1})$
    $\overline{\mathbf{H}}_t \leftarrow \Sigma + \overline{\mathbf{B}}_t$
    $\overline{\mathbf{A}}_t \leftarrow \overline{\mathbf{A}}_{t-1} + \frac{1}{t}(\mathbf{d}_t \mathbf{m}_t^\intercal - \overline{\mathbf{A}}_{t-1})$
    $\mathbf{F} \leftarrow \overline{\mathbf{A}}_t \overline{\mathbf{H}}_t^{-1}$
    $\overline{\mathbf{d}^2}_t \leftarrow \overline{\mathbf{d}^2}_{t-1} + \frac{1}{t}(\mathbf{d}_t^2 - \overline{\mathbf{d}^2}_{t-1})$
    $\psi \leftarrow \overline{\mathbf{d}^2}_t + \mathrm{sum}\left((\mathbf{F}\overline{\mathbf{H}}_t) \odot \mathbf{F} - 2\mathbf{F} \odot \overline{\mathbf{A}}_t,\ \mathrm{dim} = 1\right)$
end for
$\theta_{\mathrm{SWA}} \leftarrow \overline{\theta}_T$
return $\theta_{\mathrm{SWA}}, \mathbf{F}, \psi$

4.3 Auto-Encoding Variational Bayes

Algorithms 1 and 2 both assume that samples from the true posterior can be obtained via some separate procedure. This section describes an alternative method based on VI which learns a FA posterior directly.

4.3.1 Re-parameterisation Trick for Factor Analysis

In Section 2.3, a general VI algorithm was outlined for learning an approximate Gaussian posterior of the parameter vector of a neural network. The variational auto-encoder is based on this approach, but with a diagonal covariance matrix [4]. Suppose instead that the approximate posterior is a FA model. That is, $q(\theta) = \mathcal{N}\left(\mathbf{c}, \mathbf{F}\mathbf{F}^\intercal + \Psi\right)$

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 (2.16) becomes

$$\nabla_{\mathbf{c},\mathbf{F},\Psi}\mathbb{E}_{q(\theta)}[\log q(\theta)] - \nabla_{\mathbf{c},\mathbf{F},\Psi}\mathbb{E}_{q(\theta)}[\log p(\theta)] - \nabla_{\mathbf{c},\mathbf{F},\Psi}\mathbb{E}_{q(\theta)}[\log p(y_m|\mathbf{x}_m,\theta)], \quad (4.37)$$

where the expectation is taken over samples $\theta \sim \mathcal{N}\big(\mathbf{c}, \mathbf{F}\mathbf{F}^\intercal + \Psi\big)$. Using the re-parameterisation trick to re-write Equation (2.1), the sampling operation can be made an explicit function of $\mathbf{c}$, $\mathbf{F}$ and Ψ by setting

$$\theta = \mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z} \quad (4.38)$$

from some $\mathbf{h} \sim \mathcal{N}(\mathbf{0}^K, \mathbf{I}^{K\times K})$ and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}^D, \mathbf{I}^{D\times D})$. Using this trick, the partial derivatives in Equation (4.37) can either be evaluated exactly or approximated by Monte Carlo estimates.

4.3.2 Partial Derivatives of the Variational Distribution

Let θ be generated according to Equation (4.38). Then

$$\begin{aligned}
\log q(\theta) &= -\frac{1}{2}(\theta - \mathbf{F}\mathbf{h} - \mathbf{c})^\intercal \Psi^{-1}(\theta - \mathbf{F}\mathbf{h} - \mathbf{c}) - \frac{1}{2}\log|\Psi| - \frac{D}{2}\log 2\pi \\
&= -\frac{1}{2}(\Psi^{1/2}\mathbf{z})^\intercal \Psi^{-1}(\Psi^{1/2}\mathbf{z}) - \frac{1}{2}\log|\Psi| - \frac{D}{2}\log 2\pi \\
&= -\frac{1}{2}\mathbf{z}^\intercal(\Psi^{1/2}\Psi^{-1}\Psi^{1/2})\mathbf{z} - \frac{1}{2}\log|\Psi| - \frac{D}{2}\log 2\pi \\
&= -\frac{1}{2}\mathbf{z}^\intercal\mathbf{z} - \frac{1}{2}\log|\Psi| - \frac{D}{2}\log 2\pi.
\end{aligned} \quad (4.39)$$

Hence, $\nabla_{\mathbf{c}} \log q(\theta) = \mathbf{0}^D$, $\nabla_{\mathbf{F}} \log q(\theta) = \mathbf{0}^{D\times K}$ and

$$\begin{aligned}
\nabla_\Psi \log q(\theta) &= \nabla_\Psi\Big(-\frac{1}{2}\log|\Psi|\Big) \\
&= -\frac{1}{2}\Psi^{-1},
\end{aligned} \quad (4.40)$$

which follows from Equation (4.17). Since these derivatives do not depend on $\mathbf{h}$ and $\mathbf{z}$, and using the fact that $\nabla_{\mathbf{x}}\mathbb{E}[f(\mathbf{x})] = \mathbb{E}[\nabla_{\mathbf{x}} f(\mathbf{x})]$ for any differentiable function f,

$$\nabla_{\mathbf{c}}\mathbb{E}_{q(\theta)}[\log q(\theta)] = \mathbf{0}^D, \quad (4.41)$$

$$\nabla_{\mathbf{F}}\mathbb{E}_{q(\theta)}[\log q(\theta)] = \mathbf{0}^{D\times K}, \quad (4.42)$$

$$\nabla_\Psi\mathbb{E}_{q(\theta)}[\log q(\theta)] = -\frac{1}{2}\Psi^{-1}. \quad (4.43)$$

4.3.3 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

$$\begin{aligned}\log p(\theta) &= -\frac{1}{2}(\mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z})^\intercal \alpha\mathbf{I}(\mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z}) - \frac{1}{2}\log|\alpha^{-1}\mathbf{I}| - \frac{D}{2}\log 2\pi \\ &= -\frac{\alpha}{2}(\mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z})^\intercal(\mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z}) + \frac{n}{2}\log\alpha - \frac{D}{2}\log 2\pi.\end{aligned} \tag{4.44}$$

Hence, using the matrix calculus identities in [8],

$$\nabla_{\mathbf{c}} \log p(\theta) = -\alpha(\mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z}), \tag{4.45}$$

$$\nabla_{\mathbf{F}} \log p(\theta) = -\alpha(\mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z})\mathbf{h}^\intercal, \tag{4.46}$$

$$\nabla_{\Psi} \log p(\theta) = -\alpha(\mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z})\mathbf{z}^\intercal. \tag{4.47}$$

Since $\mathbf{h}$ and $\mathbf{z}$ are independent random variables with zero mean and unit covariance, it follows that

$$\nabla_{\mathbf{c}} \mathbb{E}_{q(\theta)}[\log p(\theta)] = -\alpha(\mathbf{F}\mathbf{h} + \mathbf{c}), \tag{4.48}$$

$$\nabla_{\mathbf{F}} \mathbb{E}_{q(\theta)}[\log p(\theta)] = -\alpha\mathbf{F}, \tag{4.49}$$

$$\nabla_{\Psi} \mathbb{E}_{q(\theta)}[\log p(\theta)] = -\alpha\Psi^{1/2}. \tag{4.50}$$

4.3.4 Partial Derivatives of the Likelihood

Let the log-likelihood, $\log p(y_m|\mathbf{x}_m, \theta)$, be parameterised by a neural network with parameters θ. Then the back-propagation algorithm [9] can be used to compute $\nabla_\theta \log p(y_m|\mathbf{x}_m, \theta)$. Since $\theta = \mathbf{F}\mathbf{h} + \mathbf{c} + \Psi^{1/2}\mathbf{z}$, it follows from the chain rule of calculus that

$$\nabla_{\mathbf{c}} \log p(y_m|\mathbf{x}_m, \theta) = \nabla_\theta \log p(y_m|\mathbf{x}_m, \theta), \tag{4.51}$$

$$\nabla_{\mathbf{F}} \log p(y_m|\mathbf{x}_m, \theta) = \nabla_\theta \log p(y_m|\mathbf{x}_m, \theta)\mathbf{h}^\intercal, \tag{4.52}$$

$$\nabla_{\Psi} \log p(y_m|\mathbf{x}_m, \theta) = \frac{1}{2}\nabla_\theta \log p(y_m|\mathbf{x}_m, \theta)\mathbf{z}^\intercal\Psi^{-1/2}. \tag{4.53}$$

Due to the functional form of a non-linear neural network, the expectation of these derivatives over $p(\theta)$ cannot be compute in closed-form and must be approximate by Monte Carlo estimates.

4.3.5 Practical Implementation

As in Algorithm 1, it is far more efficient to work with $\psi = \text{diag}(\Psi)$ rather than Ψ itself. The required partial derivatives with respect to ψ are

$$\nabla_{\psi}\mathbb{E}_{q(\theta)}[\log q(\theta)] = -\frac{1}{2}\psi^{-1}, \tag{4.54}$$

$$\nabla_{\psi}\mathbb{E}_{q(\theta)}[\log p(\theta)] = -\alpha\psi^{1/2}, \tag{4.55}$$

$$\nabla_{\psi}\log p(y_m|\mathbf{x}_m,\theta) = \frac{1}{2}\nabla_{\theta}\log p(y_m|\mathbf{x}_m,\theta)\odot\left(\mathbf{z}\odot\psi^{-1/2}\right). \tag{4.56}$$

Also, the re-parameterisation $\psi = \exp\beta$ can again be used to ensure that the variances remain positive. Similar to Equation (4.26), it then follows from the chain rule that the corresponding partial derivatives with respect to β are

$$\nabla_{\beta}\mathbb{E}_{q(\theta)}[\log q(\theta)] = -\frac{1}{2}, \tag{4.57}$$

$$\nabla_{\beta}\mathbb{E}_{q(\theta)}[\log p(\theta)] = -\alpha\psi^{3/2}, \tag{4.58}$$

$$\nabla_{\beta}\log p(y_m|\mathbf{x}_m,\theta) = \frac{1}{2}\nabla_{\theta}\log p(y_m|\mathbf{x}_m,\theta)\odot\left(\mathbf{z}\odot\psi^{1/2}\right). \tag{4.59}$$

Pseudo code for the practical implementation is given in Algorithm 3.

4.4 Factors Initialisation

One detail missing from Algorithms 1, 2 and 3 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 $\mathbf{F} = \mathbf{Q}$.

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

Algorithm 3 Auto-Encoding Variational Bayes with Factor Analysis Posterior

Input: Dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, observation dimension D, latent dimension K, learning rate η

Initialise $\mathbf{c} = \mathbf{0}^D$, $\mathbf{F} \in \mathbb{R}^{D \times K}$, $\psi = \mathbf{1}^D$
for $t = 1, \ldots, T$ **do**
  Sample a mini-batch $\mathcal{B}$ from $\mathcal{D}$
  Sample $\mathbf{h} \sim \mathcal{N}(\mathbf{0}^K, \mathbf{I}^{K \times K})$
  Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}^D, \mathbf{I}^{D \times D})$
  $\theta \leftarrow \mathbf{F}\mathbf{h} + \mathbf{c} + \psi^{1/2} \odot \mathbf{z}$
  $\mathbf{g}_\theta \leftarrow \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},y) \in \mathcal{B}} \nabla_\theta \log p(y|\mathbf{x}, \theta)$ (via back-propagation)
  $\mathbf{g}_\mathbf{c} \leftarrow \alpha(\mathbf{F}\mathbf{h} + \mathbf{c}) - \mathbf{g}_\theta$
  $\mathbf{G}_\mathbf{F} \leftarrow \alpha\mathbf{F} - \mathbf{g}_\theta \mathbf{h}^\intercal$
  $\mathbf{g}_\beta \leftarrow -\frac{1}{2} + \alpha\psi^{3/2} - \frac{1}{2} \nabla_\theta \log p(y_m|\mathbf{x}_m, \theta) \odot \left(\mathbf{z} \odot \psi^{1/2}\right)$
  $\mathbf{c} \leftarrow \mathbf{c} - \eta\mathbf{g}_\mathbf{c}$
  $\mathbf{F} \leftarrow \mathbf{F} - \eta\mathbf{G}_\mathbf{F}$
  $\beta \leftarrow \beta - \eta\mathbf{g}_\beta$
  $\psi \leftarrow \exp\beta$
end for
return $\mathbf{c}, \mathbf{F}, \psi$

Chapter 5

Online Factor Analysis Experiments

5.1 Methodology

The aim of these experiments is to test how well the online FA algorithms from Chapter 4 are able to fit observations sampled from actual FA models. The form of a FA model is given in Equation (2.3). 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 online FA algorithms (Algorithms 1 and 2) 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

$$(\mathbf{F}\mathbf{A})(\mathbf{F}\mathbf{A})^\intercal + \Psi = \mathbf{F}\mathbf{A}\mathbf{A}^\intercal\mathbf{F}^\intercal + \Psi = \mathbf{F}\mathbf{F}^\intercal + \Psi. \tag{5.1}$$

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 $\mathbf{s} > \mathbf{0} \in \mathbb{R}^D$ (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

$$\max_{i,j} \frac{|\lambda_i|}{|\lambda_j|}, \tag{5.2}$$

where the λ_i and λ_j are eigenvalues of the matrix [3]. 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 $\varepsilon \sim \mathcal{N}(\mathbf{0}, \Psi)$. The full details of how FA models were generated are given in Algorithm 4.

Algorithm 4 Generate a Factor Analysis Model

Input: Observation dimension D, latent dimension K, spectrum range $[a,b]$

1: Generate $\mathbf{c} \in \mathbb{R}^D$ by sampling entries independently from $\mathcal{N}(0,1)$
2: Generate $\mathbf{A} \in \mathbb{R}^{D\times D}$ by sampling entries independently from $\mathcal{N}(0,1)$
3: $\mathbf{M} \leftarrow \mathbf{A}\mathbf{A}^\intercal$
4: Compute the K eigenvectors, $\mathbf{v}_1, \ldots, \mathbf{v}_K \in \mathbb{R}^D$, corresponding to the K largest eigenvalues of $\mathbf{M}$
5: Construct the matrix $\mathbf{V}_K \in \mathbb{R}^{D\times K}$ with columns $\mathbf{v}_1, \ldots, \mathbf{v}_K$
6: Generate $\mathbf{s}^2 \in \mathbb{R}^D$ by sampling entries independently from $\mathcal{U}(a,b)$
7: $\mathbf{s} \leftarrow \sqrt{\mathbf{s}^2}$ (square root is applied element-wise)
8: $\mathbf{F} \leftarrow \mathbf{V}_K \odot \mathbf{s}$ (with broadcasting)
9: $s_{max} \leftarrow \max(\mathbf{s}^2)$ (largest element of $\mathbf{s}^2$)
10: Generate $\psi \in \mathbb{R}^D$ by sampling entries independently from $\mathcal{U}(0, s_{max})$
11: $\Psi \leftarrow \mathrm{diag}(\psi)$
12: **return** $\mathbf{c}, \mathbf{F}, \Psi$

Having generated a FA model, observations were sampled according to Equation (2.1). Using the data, the parameters of the model were then estimated by three sep-

arate FA algorithms: online SGA (Algorithm 1), online EM (Algorithm 2) and the scikit-learn `FactorAnalysis` estimator [7]. 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" [7]. 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 4.4, 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 (TODO: perhaps include in an appendix the derivation that Michael sent, about how this is equivalent to specifying a prior). 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.

5.2 Results and Discussion

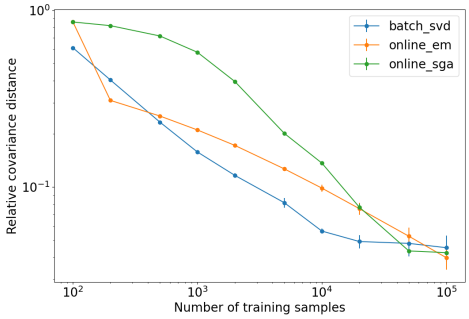
Figure 5.1 shows the relative distance between the true covariance matrix of each FA model and the corresponding estimated covariance matrix of each learning algorithm, as a function of the number of samples used to fit the approximate models. The distance between two matrices is measured by the Frobenius norm (also sometimes called the Euclidean norm) of the difference between the two matrices. The distance is then divided by the Frobenius norm of the true covariance matrix to get the relative distance.

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

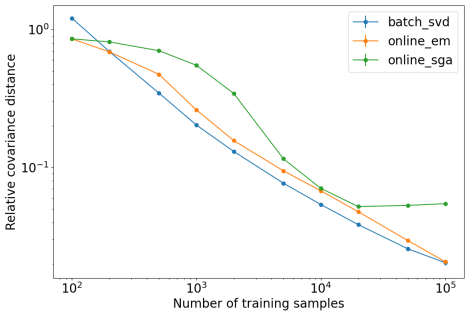
initially decreases the distance quite dramatically, even to a greater extent that batch SVD, then the rate of improvement slows before it again catches up with batch SVD at $T = 100,000$. In the third row of the figure are the results corresponding to a spectrum range of $[1, 1000]$. In this case, online 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 5.2 shows the 2-Wasserstein distance (TODO: add citation) 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. Since all algorithms are able to compute the mean of the true Gaussian distribution exactly, the Wasserstein distance in this case is just a different measure of distance between the true and estimated covariance matrices. Therefore, the shape of the plots in Figure 5.2 is very similar to the shape of the plots in Figure 5.1, albeit the y-axis scales are different.

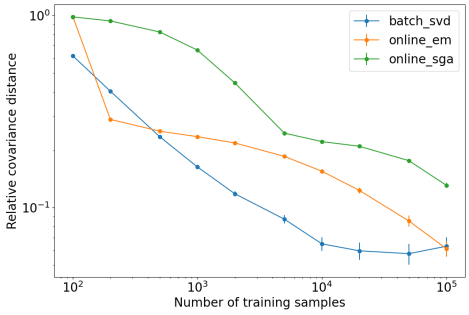
In summary, these results suggest that online EM is in general better at fitting FA models than online SGA, for all combinations of observation dimension, latent dimension, spectrum range and sample size. The SGA algorithm could possibly be improved by tuning its learning rate, or even using a learning rate scheduler. However, tuning hyperparameters is often a hassle and the fact that this is not required for online EM is another distinct advantage. 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 appears to get stuck in a local minimum at around $T = 1000$ and the rate of improvement only starts to increase again after $T = 20,000$. Perhaps online EM would be able to catch batch SVD if more than $100,000$ samples were used. However, this would be an unreasonably high number of neural network parameter vectors to sample for the use case in this thesis.



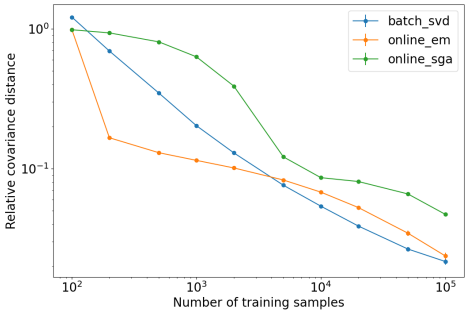
(a) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 10)$



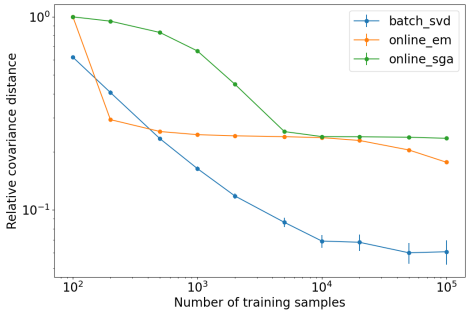
(b) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 10)$



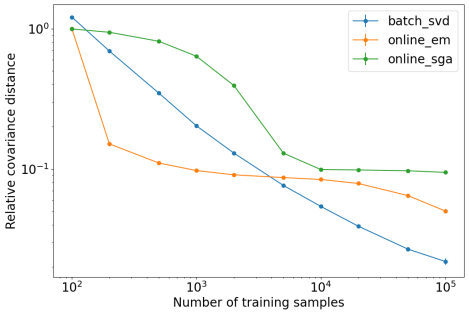
(c) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 100)$



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

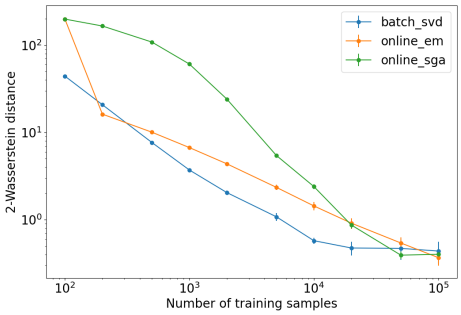


(e) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 1000)$

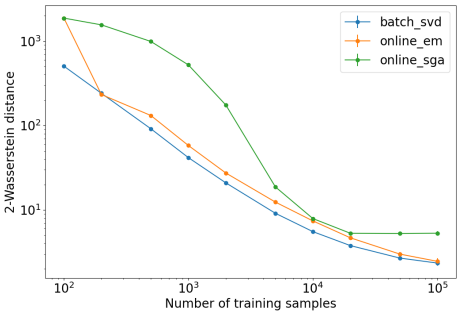


(f) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 1000)$

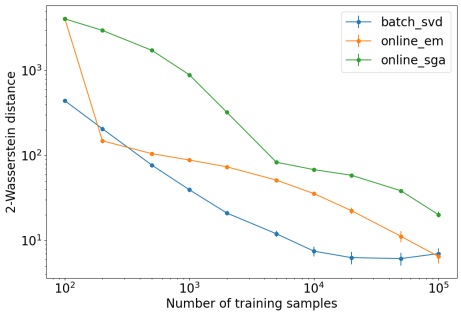
Figure 5.1: The relative distance of the estimated FA covariance matrices from the true covariance matrix as a function of the number of samples used to learn the models. The blue, orange and green lines show, for batch SVD, online EM and online SGA, respectively, the Frobenius norm of the difference 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 $\mathbf{s}^2$.



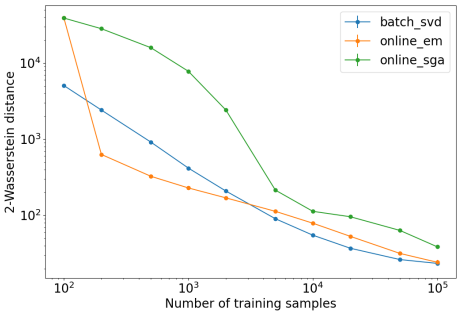
(a) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 10)$



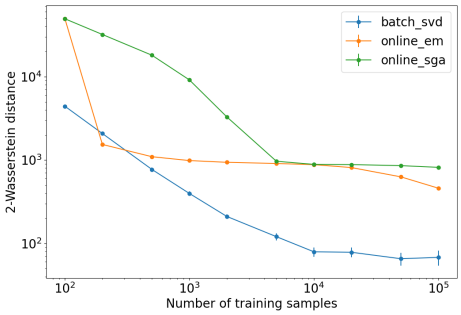
(b) $D = 1000$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 10)$



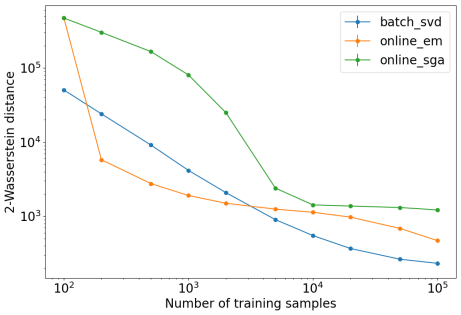
(c) $D = 100$, $K = 10$, $\mathbf{s}^2 \sim \mathcal{U}(1, 100)$



(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 5.2: The 2-Wasserstein distance between the Gaussian distribution defined by each estimated FA model and the Gaussian distribution defined by the true FA model. 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 $\mathbf{s}^2$.

Chapter 6

Linear Regression Experiments

6.1 Sampling from the Posterior

The ability of the online FA algorithms 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 the SWAG approach, 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 [5]. 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 (2.14) 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 (2.6). The precision α is a hyperparameter which controls the width of the prior $p(\theta)$. In Equation (2.10), the log-posterior of θ was written as

$$\log p(\theta|\mathcal{D}) = -\frac{\beta}{2}\sum_{n=1}^{N}\left(y_n - \theta^\intercal \mathbf{x}_n\right)^2 - \frac{\alpha}{2}\theta^\intercal\theta + \text{constant}. \tag{6.1}$$

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

$$-\frac{\beta}{2}\sum_{n=1}^{N}\left(y_n - \theta^\intercal \mathbf{x}_n\right)^2 - \frac{\alpha}{2}\theta^\intercal\theta. \tag{6.2}$$

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

$$\sum_{n=1}^{N}\left(y_n - \theta^\intercal \mathbf{x}_n\right)^2 + \frac{\alpha}{\beta}\theta^\intercal\theta. \tag{6.3}$$

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

$$\sum_{n=1}^{N} \left(y_n - \theta^\intercal \mathbf{x}_n\right)^2 + \lambda \theta^\intercal \theta \tag{6.4}$$

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 (6.4) with $\lambda = \frac{\alpha}{\beta}$ should in theory by representative of the true posterior $p(\theta|\mathcal{D})$. The purpose of the experiments in this section is to test this hypothesis.

6.1.1 Methodology

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 covariance matrix

$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}. \tag{6.5}$$

Next, the parameter vector $\theta \in \mathbb{R}^2$ was sampled from $\mathcal{N}\left(\mathbf{0}, \alpha^{-1}\mathbf{I}\right)$ with $\alpha = 0.01$. Then the outputs $y \in \mathbb{R}$ were generated according to Equation (2.4) with $\beta = 0.1$. Using this data, the true posterior in Equation (2.14) was evaluated. Before collecting samples, 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. Each experiment was 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.

6.1.2 Results and Discussion

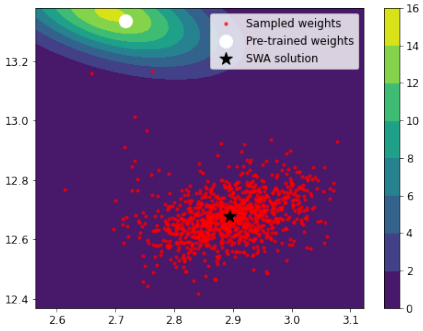
The results of each experiment are shown in Table 6.1. It is apparent from the third column that the empirical mean of the sampled parameter vectors does not match the true posterior mean when the theoretically motivated value of $\lambda = 0.1$ is used. However, if the regularisation strength is reduced 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.

Learning Rate	λ	Relative Distance from Mean	Relative Distance from Covariance	Wasserstein Distance
0.1	0.1	0.0666 ± 0.0078	0.6200 ± 0.0750	1.3480 ± 0.5087
0.1	0.001	0.0006 ± 0.0001	0.8065 ± 0.0101	0.0078 ± 0.0004
0.01	0.1	0.0620 ± 0.0070	1.7110 ± 0.6187	1.1776 ± 0.4434
0.01	0.001	0.0010 ± 0.0002	0.9870 ± 0.0042	0.0219 ± 0.0006

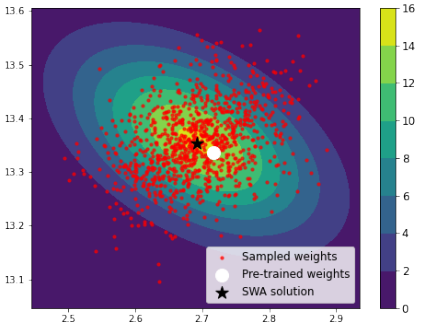
Table 6.1: Distances between the true posterior distribution of the parameter vector of a linear regression model and the empirical distribution of the parameter vectors sampled along the SGD trajectory while training the model. Relative distances between the true and empirical means and covariances 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. Each distance is the mean value over ten trials with different random seeds, and standard errors are also given.

Figure 6.1 shows, for each combination of the hyperparameters, the sampled parameter vectors for one of the ten trials in each 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

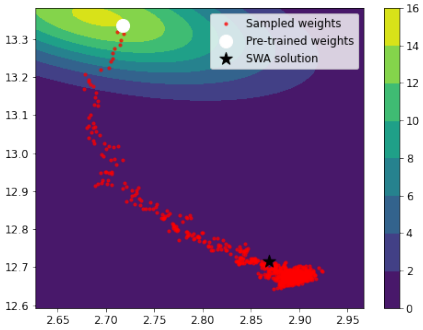
mean. However, when the learning rate is 0.1 the empirical covariance is in the wrong direction (subplot (b)), and when the learning rate is 0.01 the empirical covariance is far too narrow (subplot (d)).



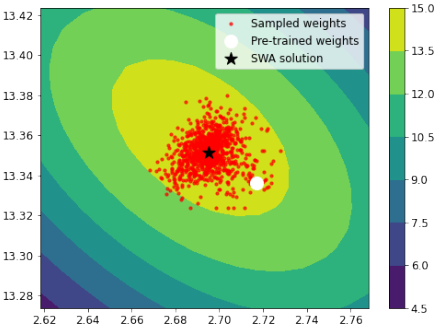
(a) Learning rate of 0.1 and $\lambda = 0.1$



(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 6.1: 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.

These results suggest that, in general, SGD does not sample parameter vectors from the true posterior distribution. 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 the 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 from the online FA

algorithms is that they fit the sampled parameter vectors well. Therefore, given the data observed in Figure 6.1, 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 in that case the posterior cannot be compute in closed form, which makes hyperparameter tuning far more difficult. These results suggest that, in general, SWAG does not actually do approximate Bayesian inference, contrary to what is claimed in [5].

It was also observed in [6] that the empirical covariance of the linear regression SGD iterates does not match the true posterior covariance. As such, the authors proposed 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 vector are then considered samples from the posterior. Figure 6.2 shows an example of such samples for the same learning rate and regularisation strength as that in Figure 6.1 (b), but with a batch size of 10 and where each red dot is the mean of 50 consecutive SGD iterates. These samples are a much better representation of the true posterior. However, to obtain such a result required 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. Some theoretical results for the "optimal" relationship between the sample size, batch size, window size and learning rate are given in [6]. However, these require knowledge of the minimum and maximum eigenvalues of the true posterior covariance, which is in general not available. Therefore, this algorithm does not seem to be a viable option in practice.

6.2 Making Predictions

Regardless of whether SWAG does Bayesian inference or not, it was shown in [5] that this method can lead to more accurate predictions. Once a Gaussian distribution has been learned from the SGD iterates, parameter vectors of the model can easily be sampled. For each sampled parameter vector, the resultant model can be used to make a different set of predictions for the test data. Final predictions are then obtained by averaging the predictions from all models. The aim of the experiments in this section is to test, for linear regression, whether this approach leads to more accurate predictions than standard SGD, when the Gaussian distribution of the parameter vectors is a FA

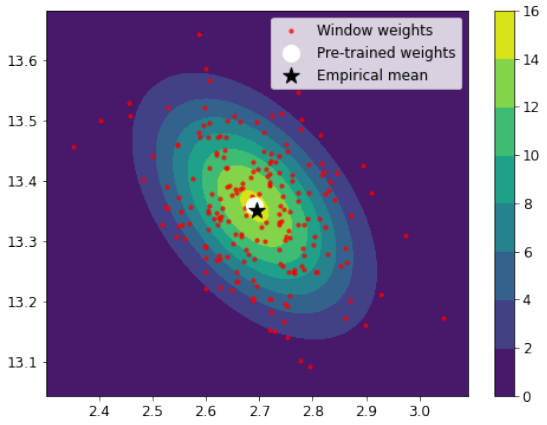


Figure 6.2: Samples constructed via finite window iterate averaging. Each red dot is the mean of the linear regression parameter vectors sampled during 50 consecutive mini-batch updates of SGD, with a batch size of 10, a learning rate of 0.1 and the regularisation strength set to 0.001. The contours 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.

model learned by either online SGA or online EM.

6.2.1 Methodology

For these experiments, four real regression datasets from the UCI Machine Learning Repository [2] were used. Namely, the Energy Efficiency[1], Boston Housing[2], Concrete Compressive Strength[3] and Yacht Hydrodynamics[4] datasets. More details about these datasets are given in Table 6.2. Note that the Energy Efficiency dataset has two target variables, heating load and cooling load, but in these experiments only the heating load was used.

For each dataset, the model was pre-trained for 500 epochs of SGD with a learning rate of 0.001, as in the synthetic experiments in Section 6.1. Then, starting from the pre-trained model, a further 100 epochs of SGD were executed. During this time, the parameter vectors sampled after each mini-batch gradient update were used to fit two

[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

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

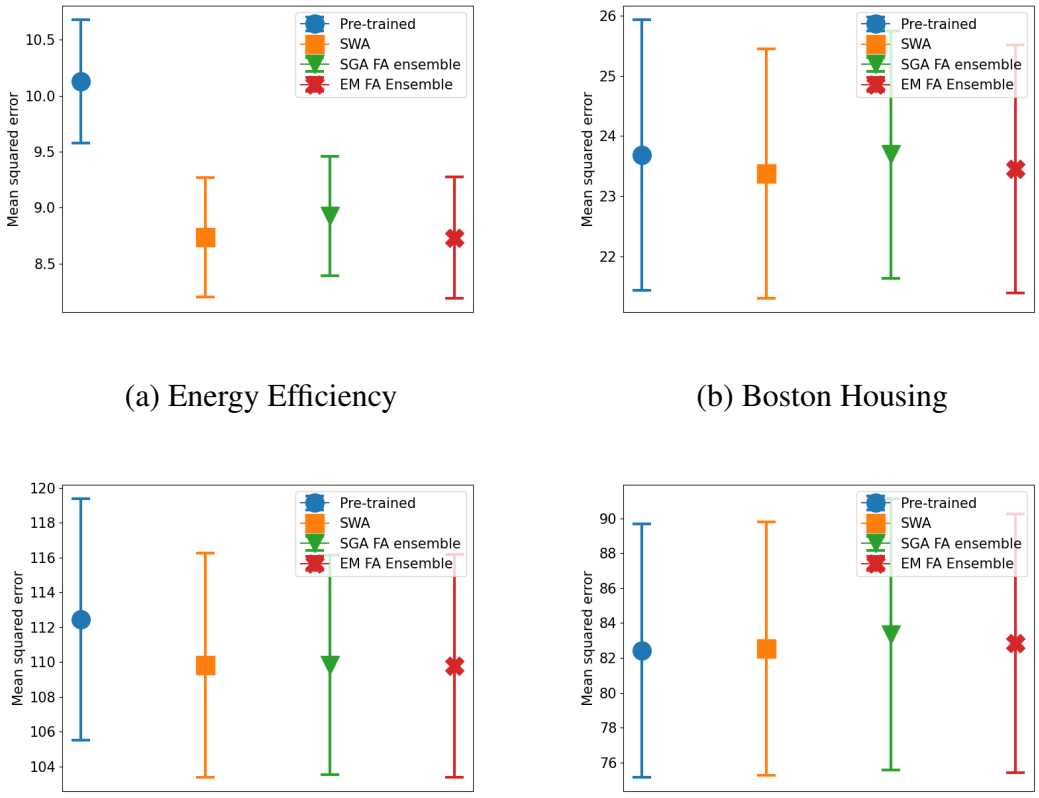
Table 6.2: UCI regression datasets used in experiments with linear regression.

FA models, one via online SGA and another via online EM. For these final 100 epochs, a learning rate of 0.1 and weight decay of 0.001 were used. These hyperparameters were chosen since it was observed in Figure 6.1 that this setting produced samples most representative of the true posterior for the synthetic data. During the pre-training phase, the weight decay was also set to 0.001 to make for a fairer comparison with the pre-trained model. For all epochs, the batch size was set to one tenth of the size of the full dataset. Both FA models had a latent dimension of three. Following the approach in the experiments in Chapter 5, the first 100 sampled parameter vectors were only used to update the running averages, $\overline{\theta}_t, \overline{\mathbf{A}}_t, \overline{\mathbf{B}}_t$ and $\overline{\mathbf{d}^2}_t$, while $\mathbf{F}$ and Ψ remained fixed. Again, the FA model updated via online SGA used a learning rate of 0.001. After training, test predictions were made using the pre-trained parameter vector, the empirical mean of the sampled parameter vectors, θ_{SWA}, and two ensembles, one for each FA model. Each ensemble was constructed by sampling 30 parameter vectors from the FA model and averaging the resultant predictions, as in the UCI regression experiments in [5]. This process was repeated for 10 folds of cross-validation. Within each fold, the input features where standardised why subtracting their mean and dividing by their standard deviation, where these statistics were calculated using the fold's training data only.

6.2.2 Results and Discussion

Figure 6.3 shows the average test mean squared error (MSE) for each dataset and prediction method. The average errors were computed over ten folds of cross-validation, and standard error bars are also plotted. Across all datasets, the SWA solution and the FA ensemble learned via online EM are the best prediction methods in general in terms of the average error. These two methods achieved almost identical results on all datasets. On three out of the four datasets they achieve lower average error than

the pre-trained solution. The difference is relatively large in the case of the Energy Efficiency dataset, but for all other datasets the standard error bars have significant overlap. Recall that, in the experiments in Chapter 5, online EM was better at fitting FA models than online SGA. This appears to translate into more accurate linear regression ensembles, as the average MSE of the online EM ensemble is less than that of the online SGA ensemble for all datasets. The numerical MSE results are also given in Table A.1 of the appendix.



(a) Energy Efficiency

(b) Boston Housing

(c) Concrete Compressive Strength

(d) Yacht Hydrodynamics

Figure 6.3: Average test mean squared error of linear regression models on four UCI regression datasets. Results are show for four different prediction methods: using the pre-trained parameter vector (blue), using the SWA solution (orange) and two ensembles formed by sampling parameter vectors from FA models fit to the SGD iterates, one via online SGA (green) and another learned via online EM (red). The markers show the average test error over ten folds of cross-validation, and standard error bars are also plotted.

These results suggest that, in the case of linear regression, fitting a FA model to

the SGD iterates via online EM and then using it to construct an ensemble can lead to more accurate predictions compared to standard SGD. However, this approach does not seem to improve on the much simpler SWA solution. Note that, in the case of linear regression, an ensemble of M parameter vectors sampled from a FA model can be written as

$$\frac{1}{M}\sum_{m=1}^{M}\theta_m^\intercal \mathbf{x} = \left(\frac{1}{M}\sum_{m=1}^{M}\theta_m^\intercal\right)\mathbf{x} = \theta_{\mathrm{FA}}^\intercal \mathbf{x}. \tag{6.6}$$

Hence, if the learned FA model is a good approximation of the empirical Gaussian distribution of the SGD iterates, θ_{FA} will be similar to θ_{SWA} for large M. This observation and the results in Figure 6.3 suggests that, if the accuracy of linear regression predictions is all that matters, the best approach is to use θ_{SWA}.

Chapter 7

Conclusions

7.1 Final Reminder

The body of your dissertation, before the references and any appendices, *must* finish by page 40. The introduction, after preliminary material, should have started on page 1.

You may not change the dissertation format (e.g., reduce the font size, change the margins, or reduce the line spacing from the default 1.5 spacing). Over length or incorrectly-formatted dissertations will not be accepted and you would have to modify your dissertation and resubmit. You cannot assume we will check your submission before the final deadline and if it requires resubmission after the deadline to conform to the page and style requirements you will be subject to the usual late penalties based on your final submission time.

Bibliography

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

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

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

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

[5] 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.

[6] 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.

[7] 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.

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

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

Appendix A

Supplementary Results

A.1 Linear Regression Predictions

Table A.1 contains the same results as Figure 6.3, but in tabular format.

Dataset	Pre-trained	SWA	SGA-FA	EM-FA
Energy Efficiency	10.13 ± 0.55	8.74 ± 0.54	8.93 ± 0.53	$\mathbf{8.73 \pm 0.54}$
Boston Housing	23.69 ± 2.25	$\mathbf{23.27 \pm 2.07}$	23.69 ± 2.06	23.46 ± 2.06
Concrete Strength	112.44 ± 6.93	109.83 ± 6.44	109.84 ± 6.30	$\mathbf{109.79 \pm 6.41}$
Yacht Hydro.	$\mathbf{82.43 \pm 7.28}$	82.54 ± 7.27	83.35 ± 7.78	82.83 ± 7.42

Table A.1: Average test mean squared error of linear regression models on four UCI regression datasets. Results are show for four different prediction methods: using the pre-trained parameter vector (second column), using the SWA solution (third column) and two ensembles formed by sampling parameter vectors from FA models fit to the SGD iterates, one via online SGA (fourth column) and another learned via online EM (fifth column). Results are average over ten folds of cross-validation, and standard errors are also given. Bold text highlights the lowest average error for each dataset, without taking into account the standard errors.