Imperial College London
Department of Computing

# Towards Deep and Distributed Bayesian Learning

Seth Nabarro

3rd November 2025

Supervised by Prof. Andrew J. Davison and Dr. Mark van der Wilk

Submitted in part fulfilment of the requirements for the degree of PhD in Computing and the Diploma of Imperial College London.

**Statement of Originality**

This thesis is my own work, except where otherwise indicated. All external sources and contributions have been appropriately referenced.

# Abstract

In machine learning, we aim to design algorithms that can discover functions from example observations. An algorithm's computational properties determine its suitability for different hardware platforms, and thus its practical utility. Despite a wide range of computational possibilities and an expanding ecosystem of hardware accelerators, modern machine learning remains largely focused on centralised execution of deep learning algorithms. In this thesis, we take a step back and explore two less prominent computational paradigms: probabilistic and distributed computation.

We first study invariance in probabilistic models and propose an alternative perspective on data augmentation in Bayesian neural networks. Unlike previous approaches, our method targets a valid likelihood function, providing a principled foundation for data augmentation and resolving questions about the "true" size of an augmented dataset used for Bayesian inference.

We then turn to distributed probabilistic learning. We argue that Gaussian belief propagation (GBP) — a message passing algorithm for probabilistic inference — has the necessary properties for efficient decentralised learning, including: i) local state updates with in-place computation, ii) exploitation of the problem structure for faster processing, and iii) robustness to asynchronicity and parallelism. We apply GBP to the distributed training of two types of model. First, in GBP Learning, we build deep probabilistic models with architectures inspired by neural networks, thus leveraging the computational benefits of GBP and the expressive power of neural networks. Second, we apply GBP to multi-robot learning, proposing a decentralised Gaussian process model in which many robots can collaboratively learn a global function using only local observation, computation and communication.

Beyond their distributed structure, the probabilistic nature of these methods endows them with several capabilities currently absent from deep learning systems, including the ability to i) learn incrementally, ii) combine learnt and hand-designed components, and iii) fuse models trained on different data.

# Contents

# List of Figures

# List of Tables

# Acronyms

**ADMM** alternating direction method of multipliers. 131

**BNN** Bayesian neural network. 29, 66, 68, 69, 71, 75, 81, 82, 85, 86, 124, 152

**BP** belief propagation. 14, 33–35, 37, 65, 90, 92, 124–128

**CNN** convolutional neural network. 16, 93, 112, 113

**CPE** cold posterior effect. 8, 66–69, 75, 77, 78, 80, 81, 85–87, 152, 154

**DA** data augmentation. 6, 8, 9, 19, 29, 36, 66–71, 73–75, 77–81, 85–87, 152, 154, 182

**DL** deep learning. 14, 16–18, 20–22, 26, 28, 29, 34, 36, 89–91, 94, 113, 124–126, 152, 153

**EBM** energy-based model. 124

**GBP** Gaussian belief propagation. 6, 34, 36, 65, 88–99, 101–104, 106, 113–116, 118, 120, 121, 124–132, 135, 139–141, 150–154

**GP** Gaussian process. 7, 9, 20, 64, 65, 129, 130, 132–136, 138–140, 143, 146, 147, 150, 151, 155

**GPU** graphics processing unit. 16

**MCMC** Markov chain Monte Carlo. 65, 124

**MLP** multi-layer perceptron. 16, 90, 102, 118

**NN** neural network. 13–16, 18–20, 28, 67, 70–72, 76, 77, 81, 88, 89, 91, 102, 118, 127, 129, 131, 144, 146, 148, 150, 151, 153, 154

**OISST** optimally interpolated sea surface temperature. 141

**PGM** probabilistic graphical model. 8, 23, 24, 65

**PSNR** peak signal-to-noise ratio. 107

**RBM** restricted Boltzmann machine. 124

**SE** standard error. 107, 114–116, 119

**SGLD** stochastic gradient Langevin dynamics. 65, 80, 85

**SST** sea surface temperature. 10, 141, 142

**TSGP** Tree-structured Gaussian process. 9, 129–136, 139, 140, 153

# Chapter 1

# Introduction

Computation can take many forms: batch or iterative, synchronous or asynchronous, centralised or distributed, sequential or parallel, and so on. Advances in hardware are constantly expanding the range of these computational modes that can be efficiently executed. Alongside conventional processors — CPUs, GPUs, TPUs [Jouppi et al., 2017], graph processors [Jia et al., 2019, Cerebras] and emerging analogue processors [Fick, 2022] — there is growing interest in less conventional platforms such as optical computing [McMahon, 2023], quantum computing [Deutsch, 1985], and even biological substrates such as slime moulds [Nakagaki et al., 2000]. Each of these may enable radically different forms of computation, with potentially profound implications for how we design and execute machine learning programs.

Despite this diversity of computational possibilities, current machine learning research is overwhelmingly focused on a single paradigm: the training of deep neural networks (NNs) by backpropagation on centralised hardware, typically GPUs or tightly connected GPU clusters. This narrow focus is striking given the rich landscape of alternative computational styles.

In this thesis we explore two less prevalent modes of computation for machine learning: distributed and probabilistic algorithms. Distributed computation aims to decompose large problems into smaller tasks which can be executed on different processors without central coordination. Such algorithms often have less restrictive data dependencies, admitting greater parallelisation. In probabilistic inference, we use computation to reason about uncertainty, updating beliefs in light of new data. At the intersection of these two modes is belief propagation [BP; Pearl, 1982], in which local nodes compute and exchange probabilistic

messages which are collated to estimate posterior beliefs. BP does computation *in place*, i.e. where the data are, thus avoiding common bottlenecks associated with moving data between storage and processing.

This introduction proceeds as follows. We first review the current state of machine learning, highlighting its successes and shortcomings, before discussing the Bayesian perspective and where this may be beneficial. We include some analysis on the relation between Bayesian inference and computation, and conclude with a summary of the contributions made in the remainder of the thesis.

## 1.1 Machine Learning

The central aim of machine learning is to deduce *procedures* from *observations*. Procedures may come in the form of functions which categorise, forecast or generate; they may constitute policies which enable an agent to navigate an environment. In general, they are statistical models with parameters $\theta$, and learning is the process of identifying appropriate values of $\theta$ such that the statistical model performs well on the available training data. This is quantified by a *loss function*, $\mathcal{L}(\theta)$ for which low values indicate good performance, so learning can be cast as the optimisation problem

$$\theta^* = \text{argmin}_\theta \mathcal{L}(\theta). \tag{1.1}$$

Given $\theta^*$, the model may then be used to process other data not seen during training. This raises the question of *generalisation*: has the model *overfit* to the training data, or can it also perform well on new inputs?

While numerous styles of machine learning algorithm have been proposed, the recent boom has largely been driven by progress in *deep learning* (DL). We will now review this subfield.

### 1.1.1 Deep Learning

The core idea of DL [Goodfellow et al., 2016] is to build a flexible function $f(\cdot)$ through composition, connecting many component functions or *layers*, $\{f_i(\cdot)\}_{i=1}^L$ together, to form a "deep" NN

$$f = f_L \circ \ldots \circ f_2 \circ f_1. \tag{1.2}$$

Each layer has parameters $f_i(\cdot) = f_i(\cdot;\theta_i)$, and we denote the concatenation of parameters for all layers as $\theta$, so we can write $f(\cdot) = f(\cdot;\theta)$. The input $\mathbf{x}$ is fed through the sequence of layers, computing the intermediate *hidden activations* at each step $\{\mathbf{h}_l\}_{l=1}^{L-1}$ before reaching the output $\hat{\mathbf{y}}$

$$\mathbf{h}_1 = f_1(\mathbf{x};\theta_1) \tag{1.3}$$

$$\mathbf{h}_2 = f_2(\mathbf{h}_1;\theta_2) \tag{1.4}$$

$$\vdots$$

$$\hat{\mathbf{y}} = f_L(\mathbf{h}_{L-1};\theta_L). \tag{1.5}$$

Many approaches exist for training NNs, but the predominant method is the *backpropagation* algorithm, often abbreviated to "backprop" [Werbos, 1974, Rumelhart et al., 1986]. Backprop computes the exact *gradient* of a chosen loss function $\mathcal{L}(\cdot,\cdot)$ with respect to the parameters of the network $\theta$. This gradient $\partial\mathcal{L}/\partial\theta$ is the direction of $\theta$ for which $\mathcal{L}$ increases most steeply.

The algorithm proceeds as follows. In the *forward pass*, the network is queried with a training input $\mathbf{x}_i$ to generate a prediction $\hat{\mathbf{y}}_i$. This prediction is compared to the observed target $\mathbf{y}_i$ under a chosen loss function $\mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i) =: \mathcal{L}_i$. In the *backward pass*, the the gradient of the loss with respect to the parameters of each layer $\theta_l$ is computed by unrolling, layer-by-layer, through the network according to the chain rule of derivatives. This begins at layer $L$, and works back to layer 1. The gradient at an arbitrary layer $l$ is given by

$$\frac{\partial\mathcal{L}_i}{\partial\theta_l} = \frac{\partial\mathcal{L}_i}{\partial\hat{\mathbf{y}}_i}\frac{\partial\hat{\mathbf{y}}_i}{\partial\mathbf{h}_{L-1,i}}\frac{\partial\mathbf{h}_{L-1,i}}{\partial\mathbf{h}_{L-2,i}}\cdots\frac{\partial\mathbf{h}_{l+1,i}}{\partial\mathbf{h}_{l,i}}\frac{\partial\mathbf{h}_{l,i}}{\partial\theta_l}. \tag{1.6}$$

The gradients for the parameters of all layers $\{\partial\mathcal{L}_i/\partial\theta_l\}_{l=1}^{L}$ are concatenated to give $\partial\mathcal{L}_i/\partial\theta$. While backpropagation is a means to compute the gradient of the loss function, it does not specify how the gradient should be used. A simple and effective method is to use *gradient descent* – update $\theta$ by taking a step in the direction of the negative gradient

$$\theta \leftarrow \theta - \alpha\frac{\partial\mathcal{L}_i}{\partial\theta}, \tag{1.7}$$

where the *learning rate* $\alpha$ is a hyperparameter which controls the stepsize. Intuitively, repeating this procedure iteratively moves the parameters in the direction which reduces

the loss function, or increases the accuracy of the network, in an attempt to solve (1.1).

#### 1.1.1.1 The Deep Learning Revolution

The foundations of DL go back to early models of biological neural activity. McCulloch and Pitts [1943] introduced a simple threshold model for a biological neuron with fixed weights and binary inputs. This was extended by Rosenblatt [1958] who designed the *Perceptron*: a NN with a single learnable layer. NNs later fell out of favour after Minsky and Papert [1969] showed they were fundamentally unable to learn non-linear decision boundaries, and therefore limited in the complexity of problems they could solve. It was not until this restriction was addressed with the advent of multi-layer perceptrons (MLPs) trained by backpropagation [Werbos, 1974, Rumelhart et al., 1986] that DL regained some popularity. The introduction of convolutional neural networks [CNNs; Fukushima, 1980, LeCun et al., 1989] then enabled effective learning with image data. Despite these breakthroughs, progress was hampered by the issue of vanishing and exploding gradients, which was addressed by i) Glorot and Bengio [2010] who proposed a more effective method for initialising NN parameters and ii) Glorot et al. [2011] who argued for non-saturating activation functions such as ReLU.

The first major breakthrough came with the *AlexNet Moment* of 2012 where a CNN achieved record performance for image classification and object localisation tasks on the ImageNet dataset [Russakovsky et al., 2015]. Krizhevsky et al. [2012] trained a (at the time) large CNN on the (at the time) large ImageNet dataset [Deng et al., 2009], using an efficient implementation with low-level code for graphics processing unit (GPU) acceleration. The model far outperformed other entries in the ImageNet competition. AlexNet arguably paved the way for DL-based computer vision, which has now become dominant, achieving state-of-the-art in many computer vision tasks since. A non-exhaustive list of tasks e.g. semantic segmentation [Farabet et al., 2012], object detection [Girshick et al., 2014], image captioning [Karpathy and Fei-Fei, 2015, Vinyals et al., 2015], image generation [Brock et al., 2018, Rombach et al., 2022] and optical flow [Teed and Deng, 2020].

Another major breakthrough came when Vaswani et al. [2017] introduced the *Transformer* network architecture. The transformer is based on the concept of *self-attention*, a mechanism for computing the pairwise relevance between units of information (tokens) at different locations in a sequence. Unlike prior recurrent neural sequence models [e.g. Sutskever et al., 2014], attention can be parallelised over the sequence and executed efficiently on GPU. It

drastically improved the state-of-the-art performance across many language modelling tasks [Islam et al., 2024] and forms the central component of all popular language models today [Gemini Team et al., 2023, Dubey et al., 2024, Lindsey et al., 2025]. Beyond language, they are now recognised as powerful, general purpose sequence models and are commonly used across many domains including computer vision [Dosovitskiy et al., 2020, Dehghani et al., 2023], biological modelling [Abramson et al., 2024, Rives et al., 2021], weather modelling [Price et al., 2025] and finance [Mishev et al., 2020].

In 2024, the Nobel prizes for both Physics and Chemistry were awarded for work either in the development of DL [Nobel Committee for Physics, 2024] or in its application [Nobel Committee for Chemistry, 2024]. This is a clear testament to its impact.

#### 1.1.1.2 What Makes Deep Learning Successful?

We now reflect on the features of DL which have contributed to its widespread success. Below is a shortlist of points relevant to this thesis, but we note there are many others.

1. **A Suitable Hardware Platform.** Part of the reason for AlexNet's [Krizhevsky et al., 2012] was the then recent availability of large, labelled datasets such as ImageNet [Deng et al., 2009]. However, scaling to this size of training data required a hardware platform which could effectively accelerate DL. Krizhevsky et al. [2012] were some of the early adopters of using Graphics Processing Units (GPUs) for this purpose, using NVIDIA's CUDA platform which allowed them to be programmed for non-graphics applications. GPUs use a same instruction multiple data (SIMD) style of parallel computation, which is well suited to accelerating the matrix multiplications central to DL.

   The *Bitter Lesson* [Sutton, 2019] of machine learning is that approaches which are better able to make use of computation are more successful than carefully designed approaches based on human-knowledge priors. Similarly, Hooker [2021] note that algorithms often succeed or fail not because of their intrinsic merit, but because of the availability (or lack) of a suitable hardware and software platform. They term this the *Hardware Lottery*. That Krizhevsky et al. [2012], whose simple algorithm scaled to a large model on GPUs, far outperformed other entrants with more elaborate algorithms [Russakovsky et al., 2015] is a typical illustration of both the bitter lesson and the hardware lottery.

The past decade and a half has seen a dramatic increase in the availability and power of GPUs and similar processors. Further, GPU architectures have been optimised for DL in addition to graphics, with mixed-precision support and NVIDIA Tensor Cores for example. This increase in availability and efficiency has made GPUs the default processor for DL, and has undoubtedly skewed the direction of machine learning research towards models which run efficiently on GPU [Hooker, 2021].

2. **Availability of Webscale Data.** While AlexNet was trained on the ImageNet dataset [Deng et al., 2009], the largest labelled dataset at the time, we have since seen a continuous growth in the size of training sets. In particular, unlabelled datasets gathered by web-scraping have grown considerably, and generative pre-training [GPT Radford et al., 2018] – masking out some of the input and training a network to "fill in the blanks" – has provided a means to leverage them. Transformer models trained with GPT on huge datasets are the basis of today's frontier models.

3. **Large Models and Overparameterisation.** Empirical analysis has found that overparameterised NNs – those with far more parameters than training examples – often outperform smaller networks [Kaplan et al., 2020]. Further, some theoretical works have derived generalisation bounds which imply performance improves in overparameterised regimes [Neyshabur et al., 2018, Allen-Zhu et al., 2019]. Indeed, the training of such large, overparameterised networks has become the standard choice, largely enabled by growth in available compute.

4. **Good Inductive Biases.** While deep NNs are highly flexible models which can fit a wide range of datasets, the best performing DL systems are those with suitable *inductive bias* for a given task. That is, systems which have a preference towards learning functions which generalise well. Indeed, MLPs with one hidden layer are theoretically able to approximate *any* function (in the limit of infinite width) [Hornik et al., 1989], but it has been observed in practice that non-fully connected layers, with stronger inductive bias, perform better.

   Invariance or equivariance is a common type of inductive bias. An invariant function is one which is unaffected by a transformation $\tau(\cdot)$ applied to the input,

$$f(\mathbf{x}) = f(\tau(\mathbf{x})), \tag{1.8}$$

where an equivariant function is one for which the transformation and the function commute

$$\tau\left(f\left(\mathbf{x}\right)\right) = f\left(\tau\left(\mathbf{x}\right)\right). \tag{1.9}$$

If it is known *a-priori* that a particular mapping is invariant (or equivariant), then designing a learning algorithm which guarantees this will result in better generalisation. Consider, for example, a computer vision system which classifies different types of plant. It is clear that the predicted type of plant should not depend on whether the camera is tilted, upright or upside-down when the photo is taken, so it would be sensible to make this system invariant to rotation. This would enable a model trained only on landscape-oriented images to predict accurately with portrait-oriented images.

Many neural architectures are designed to be invariant or equivariant [Shawe-Taylor, 1993]. For example, convolutional layers [LeCun et al., 1989, Fukushima, 1980] are translation equivariant, meaning the same function is applied in each local region of the input image. Thus when e.g. classifying an image according to the presence of an object type, it does not matter *where* in the image the object occurs, a translation-invariant model will give the same prediction. Another example is permutation equivariance. The self-attention mechanism behind transformers [Vaswani et al., 2017] is equivariant to shuffling the order (permuting) of elements in the input sequence. Moreover, graph NNs are generally permutation invariant or equivariant over the nodes in the graph [Keriven and Peyré, 2019].

However, designing architectures which are invariant to some transformations or combinations of transformations may be challenging, so a common approach is to use *data augmentation* (DA). The idea is to create additional training examples by applying the transformation to the input and either applying the same transformation to the output target (in the case of equivariance) or leaving the output target unchanged (for invariance). This simple procedure has been successful in improving performance in a wide variety of machine learning applications [see Loosli et al., 2007, Krizhevsky et al., 2012, Bishop, 2006, for example], and recent work has analysed the effect of DA on invariances in the learned functions [Dao et al., 2019, Chen et al., 2020, Lyle et al., 2020].

Note that the invariance or equivariance is not enforced by model construction (as with architectural design) but is encouraged by including the transformed examples in

the dataset and hoping the network learns the invariance or equivariance. Van der Wilk et al. [2018] formalise the idea of an approximate invariance in the notion of *insensitivity*: if a predicted output $f(\cdot)$ is unlikely to differ significantly between the original input $\mathbf{x}$ and an augmented input $\tau(\mathbf{x})$, it can be said to be *insensitive* to the augmentation distribution. That is,

$$p\left(|f(\mathbf{x}) - f(\tau(\mathbf{x}))|^2 > \delta\right) < \epsilon, \quad \tau \sim p(\tau) \tag{1.10}$$

where $p(\tau)$ is the distribution over possible augmentation transformations, $\delta$ is a threshold difference in function output, and $\epsilon$ is a threshold maximum probability.

#### 1.1.1.3 The Limitations of Deep Learning

Having discussed the merits of DL, we now summarise its current limitations.

1. **Data Inefficiency.** Large NNs are highly expressive models which, in the limit of a large dataset, often achieve state-of-the-art performance. However, they generally perform poorly when only a small training set is available [Lake et al., 2015, D'Amario et al., 2022, Kaplan et al., 2020]. This is in stark contrast to biological learning, in which a child need only observe one or a few examples of a concept to recognise other instances. Further, many other machine learning models such as Gaussian processs (GPs) or random forests are known to be more sample efficient [Goldin et al., 2023, Welbl, 2014] though do not have the same asymptotic performance as NNs.

2. **Inability to Learn Incrementally.** DL must be done with a clear distinction between training and testing. At training time, weights are updated to minimise the loss, and after training they are fixed. However, even during training, the resultant model is sensitive to the ordering of the training examples. In particular, later training examples which contain new concepts, may cause the NN to "forget" concepts learnt previously.

   This phenomenon is known as *catastrophic forgetting*, indicating how strongly previous learnings are overwritten [McCloskey and Cohen, 1989, Ratcliff, 1990, French, 1999]. NNs are thus unable to learn incrementally, as biological organisms do, from streams of data. Instead, a common approach is to collect a *replay buffer* of examples from the stream of experience, and occasionally retrain the model on these examples to prevent

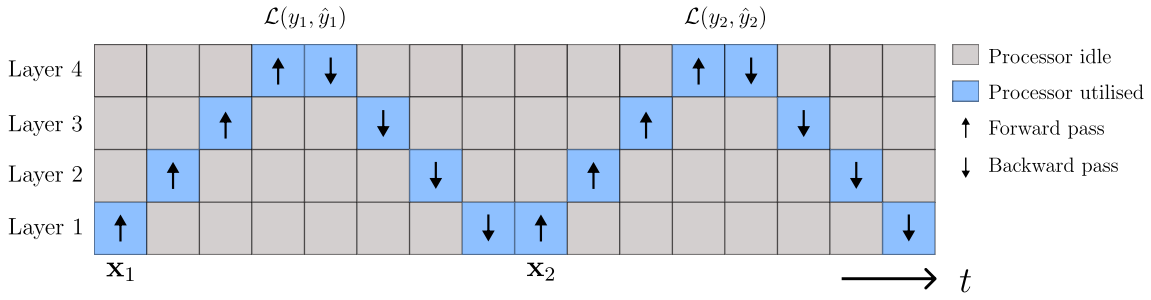



Figure 1.1: **Forward and backward locking with backpropagation.** The sequential nature of the backpropagation algorithm restricts model parallelism and limits hardware utilisation. This example shows the schedule of a four layer network processing two batches $(\mathbf{x}_1, \hat{y}_1), (\mathbf{x}_2, \hat{y}_2)$.

their content being forgotten [Lin, 1992, Mnih et al., 2013]. This incurs significant memory cost, and introduces additional hyperparameters: how should we select which examples are added to the buffer? How often should we train on the replay buffer vs the online data? How many replay buffer update steps should we use? and so on. Despite catastrophic forgetting in DL being observed as early as 1989 [McCloskey and Cohen, 1989], continual DL remains an active research area today [Lyle et al., 2023, van de Ven et al., 2024, Li et al., 2024, Zhai et al., 2024], indicating it remains an open problem.

3. **Lack of Model Parallelism.** In many ways, DL is highly scalable: training is of linear complexity in the dataset size and commonly used network architectures are highly efficient on GPUs with optimised hardware designs and low-level code. Further, training can be easily parallelised over many processors by replicating the model, computing the gradients for different batches on different processors, then aggregating the computed gradients. This is known as *data parallelism* [Dean et al., 2012, Andersen and Park, 2014].

   However, within a single model, backpropagation is inherently sequential. The forward pass proceeds from layers $f_1$ to $f_L$, the loss is calculated and gradients are computed from $\partial\mathcal{L}/\partial\theta_L$ back to $\partial\mathcal{L}/\partial\theta_1$ (see Figure 1.1). For layer $l$, the forward computation depends on the output of $l-1$ and the backwards pass depends on $\partial\mathbf{h}_{l+2}/\partial\mathbf{h}_{l+1}$. Thus after their part of the forward pass, the early layers e.g. $l = 1$ are sat idle waiting for the forward pass for 2 to $L$ to complete, and the backwards error signal from $L$ back

to 1, before they can compute the gradient of their parameters.

This is known as the *forward* and *backward locking*, and prevents the parallelisation of the backpropagation algorithm over the layers of the network. The result is poor hardware utilisation when a model is spread over multiple processors, as illustrated by the prevalence of grey squares in Figure 1.1. While mitigation strategies exist, such as model pipelining which processes different batches with different layers simultaneously [Huang et al., 2019], they require considerable engineering to make efficient, incur additional memory to store the activations for many batches, and result in large effective batch sizes which can be detrimental to generalisation [Lei et al., 2018, Masters and Luschi, 2018, Marek et al., 2025].

4. **The von Neumann Bottleneck.** As discussed in the strengths (point 1), GPUs have been highly effective accelerators of DL and have significantly improved in power and efficiency over the past $\sim 20$ years. However, we argue they will eventually reach a fundamental limit due to the separation of memory and computation.

   As memory located at the GPU processing nodes is minimal, constant data movement between the large off-chip DRAM and the processors is necessary. The bandwidth of this memory is therefore the bottleneck to efficient execution, and significant time and energy is spent not in processing but moving data around Gholami et al. [2024], Delestrac et al. [2024]. With Moore's law [Moore, 1998] coming to an end and Dennard scaling [Dennard et al., 2003] over, it is unlikely that this bottleneck will be solved without a significantly different accelerator design. Even with processors that *do* have significant local memory, DL would still be limited by the bandwidth *between* cores. This is because, with backprop, there is limited local computation that can be executed before communication is necessary (e.g. due to point 3).

## 1.2 A Bayesian Perspective

We now present a high-level overview of Bayesian modelling and inference, as well as their application to machine learning. We then discuss how Bayesian approaches might address some of the shortcomings of today's DL systems, and consider the computational properties of Bayesian inference.

### 1.2.1 Bayesian Inference and Modelling

In the Bayesian approach to statistical modelling, probabilities represent subjective states of belief which are updated in light of observations [Jaynes, 2003]. Broadly, the modelling and inference can be broken down into three stages:

1. **Definition of a Generative Model.** We first define a model which encodes our assumptions about how the data were generated. These assumptions are statements about the independence of certain variables, the hierarchy between dependent variables, and the statistical distributions which parameterise their relationships. This generative model is a joint probability distribution over all variables, and the way that we factorise this joint distribution reflects our assumptions, for example

$$p(a,b,c) = p(a)\,p(b|a)\,p(c|a) \tag{1.11}$$

says that $b$ and $c$ are independent if $a$ is known, where

$$p(a,b,c) = p(a)\,p(b)\,p(c|a,b) \tag{1.12}$$

says that $a$ and $b$ are independent, but become dependent if $c$ is known.

A convenient way of writing a joint distribution is as a *probabilistic graphical model* (PGM). This is an intuitive representation of its independence structure. The variables in the model are nodes in the graph, and connections between two variables indicate that both feature in one of the terms of the joint distribution. We present the Bayesian networks – directed PGMs – for (1.11) and (1.12) in Figures 1.2a and 1.2b.

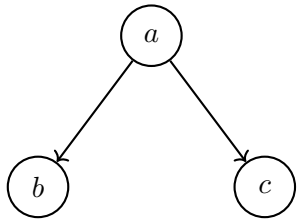
The generative model can also be specified in an *undirected* form, as the product of functions rather than conditional probabilities. For example, we may write (1.11) as

$$p(a,b,c) = \frac{1}{Z}\phi_1(a)\,\phi_2(a,b)\,\phi_3(a,c)\,, \tag{1.13}$$

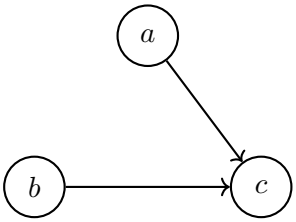
and (1.12) as

$$p(a,b,c) = \frac{1}{Z}\phi_1(a)\,\phi_2(b)\,\phi_3(a,b,c) \tag{1.14}$$

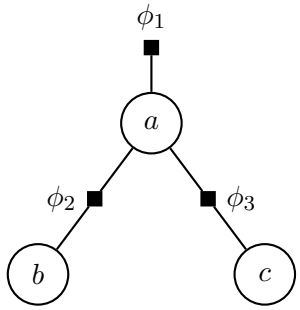
where $Z$ is the normalising constant, also known as the partition function. Undirected graphical models can be illustrated as factor graphs (see Figures 1.2c and 1.2d).



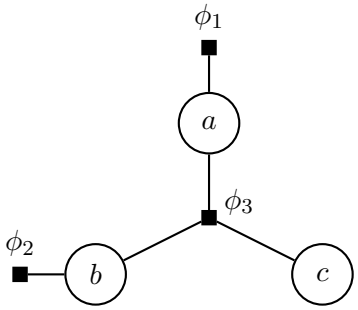

(a) Bayesian network for (1.11).




(b) Bayesian network for (1.12).




(c) Factor graph for (1.11).




(d) Factor graph for (1.12).

Figure 1.2: **Examples of PGMs over three variables:** $a$**,** $b$ **and** $c$. Arrows represent conditional terms $p(x|y)$ in the Bayesian networks, where squares represent factors – non-negative functions of the connected variables.

Variable nodes (circles) are connected to the factors (squares) which depend on them.

2. **Marginalisation of Nuisance Variables:** Variables in the model which we do not observe and are not interested in inferring are known as nuisance variables. They should be *marginalised* out of the model, i.e. we should average the model over the values we believe these variables could take. For example, for a generative model $p(a, b, c)$, with nuisance variable $c$, can be marginalised

$$p(a, b) = \sum_c p(a, b, c). \tag{1.15}$$

3. **Conditioning on Observations.** Given the assumed generative model, and observations of a subset of its variables we then ask *what is my updated belief about one/some/all of the unobserved variables?*. In the Bayesian framework, the answer is provided by the rule of conditional probability, which gives rise to *Bayes rule*. Suppose we have a model of variables **a** and **b**, and we observe **b**. Our *posterior* belief about **a**

is

$$p(\mathbf{a}|\mathbf{b}) = \frac{p(\mathbf{a},\mathbf{b})}{p(\mathbf{b})} \tag{1.16}$$

$$= \frac{p(\mathbf{b}|\mathbf{a})\,p(\mathbf{a})}{p(\mathbf{b})} \tag{1.17}$$

where the second equality (Bayes rule) comes from the chain rule of probability. In this case, $p(\mathbf{a},\mathbf{b})$ is the generative model we condition.

Bayes rule formalises the updating of a belief with observations. $p(\mathbf{a})$ is our *prior* distribution on $\mathbf{a}$, which should encode everything we know about it before we make observations $\mathbf{b}$. The *likelihood* $p(\mathbf{b}|\mathbf{a})$ tells us which values of $\mathbf{a}$ are consistent with the observations; it "cuts away" regions of $\mathbf{a}$ for which $p(\mathbf{a}) > 0$ but that are not predictive of observations $\mathbf{b}$. The *marginal likelihood* $p(\mathbf{b}) = \sum_{\mathbf{a}} p(\mathbf{b}|\mathbf{a})\,p(\mathbf{a})$ normalises the numerator to ensure the posterior is a valid probability distribution.

### 1.2.2 Bayesian Machine Learning

Broadly speaking, Bayesian machine learning is the set of methods which apply the recipe in Section 1.2.1 to the problem of machine learning. For a model with parameters $\theta$, training data $\mathcal{D}$ and latent variables $U$, we have a generative model $p(\mathcal{D},U,\theta)$, and we seek the posterior over parameters

$$p(\theta|\mathcal{D}) = \frac{\sum_U p(\mathcal{D},U,\theta)}{p(\mathcal{D})} \tag{1.18}$$

$$= \frac{\sum_U p(\mathcal{D}|U,\theta)\,p(\theta)\,p(U)}{p(\mathcal{D})}. \tag{1.19}$$

Predictions on new examples $\mathcal{D}^*$, can be generated by averaging over this posterior, giving the *Bayesian model average*

$$p(\mathcal{D}^*|\mathcal{D}) = \int p(\mathcal{D}^*|\theta)\,p(\theta|\mathcal{D})\,\mathrm{d}\theta. \tag{1.20}$$

#### 1.2.2.1 Why Bayes?

What are the benefits of this Bayesian approach to machine learning over the standard, loss-minimisation method (1.1)? We provide a non-exhaustive list below.

1. **Better Accuracy and Greater Sample Efficiency.** For a model with many parameters relative to the training set size, the parameter values may be underconstrained. In this regime, accurate Bayesian inference will result in a posterior $p(\theta|\mathcal{D})$ which has support over a range of $\theta$. By accounting for all of these model fits when making predictions (1.20), we are likely to predict more accurately. Furthermore, in the "small data" regime, the incorporation of a sensible prior $p(\theta)$ may bias the posterior towards values of $\theta$ which generalise well. In contrast, the loss-minimisation approach (1.1) will converge to a single point estimate of $\theta$, and may overfit to a small dataset, thus generating poor predictions.

2. **Calibrated Uncertainty.** Similarly, if we believe a variety of $\theta$ values are possible given the data, then generating and averaging predictions for all of these values should yield $p(\mathcal{D}^*|\mathcal{D})$ which reflects this uncertainty. Accurate prediction uncertainty enables the model to better interface with external systems – decision engines, humans, other models. We elucidate briefly below.

   Uncertainty calibration is important for effective downstream decision-making [Berger, 2013]. In Bayesian decision theory, an observation maps to a decision via a *decision rule*, and there is a penalty based on how the decision and the true state vary. In this case, the estimated risk is the penalty of an action averaged over the predicted distribution of true states. This may be a poor estimate of risk if the distribution is not well calibrated. Further, in safety critical applications such as medical diagnosis, autonomous driving or industrial automation, accurate uncertainty quantification may enable better computer-human collaboration, ensuring decisions based on highly uncertain predictions are made by humans.

   Many systems need to fuse information from multiple sources, one of which may be a model's prediction. In this case the model uncertainties control the weighting of prediction relative to other sources. This is common in robotics, where estimation of a robot's state (e.g. pose) may depend on a combination of noisy sensor measurements and model predictions. Over-confident predictions will lead to an estimated state which is too close to the model prediction, and vice versa.

3. **A Natural Way to Learn Continually.** While the standard DL paradigm suffers from catastrophic forgetting, we argue that effective incremental learning is inherent in Bayes rule.

Let us consider how the posterior for a stream of training data $\mathcal{D} = [\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T]$ decomposes. We will assume independent likelihood terms for each example: $p(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T|\theta) = \prod_t p(\mathcal{D}_t|\theta)$, and for simplicity consider a model without additional latent variables $U$. Starting at (1.19),

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)\, p(\theta)}{p(\mathcal{D})} \tag{1.21}$$

$$= \frac{p(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T|\theta)\, p(\theta)}{p(\mathcal{D})} \tag{1.22}$$

$$= \frac{p(\mathcal{D}_1|\theta)\, p(\mathcal{D}_2|\theta) \ldots p(\mathcal{D}_T|\theta)\, p(\theta)}{p(\mathcal{D})} \tag{1.23}$$

as per the independent likelihood assumption. Let us unroll the denominator according to the chain rule of probability $p(\mathcal{D}) = p(\mathcal{D}_1) \prod_{t=2}^{T} p(\mathcal{D}_t|\mathcal{D}_1, \ldots, \mathcal{D}_{t-1})$

$$p(\theta|\mathcal{D}) = \underbrace{\underbrace{\underbrace{\frac{p(\mathcal{D}_1|\theta)\, p(\theta)}{p(\mathcal{D}_1)}}_{p(\theta|\mathcal{D}_1)} \frac{p(\mathcal{D}_2|\theta)}{p(\mathcal{D}_2|\mathcal{D}_1)}}_{p(\theta|\mathcal{D}_1, \mathcal{D}_2)} \cdots \frac{p(\mathcal{D}_T|\theta)}{p(\mathcal{D}_T|\mathcal{D}_1, \ldots, \mathcal{D}_{T-1})}}_{p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_T)}. \tag{1.24}$$

The denominator at step $t$ can be written

$$p(\mathcal{D}_t|\mathcal{D}_1, \ldots, \mathcal{D}_{t-1}) = \int p(\mathcal{D}_t|\mathcal{D}_1, \ldots, \mathcal{D}_{t-1}, \theta)\, p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_{t-1})\, \mathrm{d}\theta \tag{1.25}$$

$$= \int p(\mathcal{D}_t|\theta)\, p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_{t-1})\, \mathrm{d}\theta, \tag{1.26}$$

due to the independent likelihood assumption. As both the numerator and the denominator at each stage require only the likelihood of the current data $p(\mathcal{D}_t|\theta)$, and the posterior from previous observations $p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_{t-1})$, we can arrive at the same final posterior by updating our posterior one datapoint at a time: condition on $\mathcal{D}_1$ to get $p(\theta|\mathcal{D}_1)$, use this as a prior when conditioning on $\mathcal{D}_2$ and repeat for $t = 3, \ldots, T$. We emphasise that the only assumption we have used is that the likelihood of each example is independent of the others.

In practice, the batch posterior and the online posterior may vary due to compounding of inference and approximation errors over the sequence. Nonetheless, that Bayesian

inference is theoretically agnostic to batch vs online updating implies it is well-suited to continual learning.

#### 1.2.2.2 Gaussian Processes

Gaussian processes [GPs; Williams and Rasmussen, 2006] are distributions in the space of functions which admit supervised learning via Bayesian inference. Prior knowledge can be encoded in a *covariance function*, which restricts the space of possible functions to be e.g. smooth, periodic, linear. For observations with Gaussian likelihoods, one can compute the GP posterior over functions exactly.

GPs are highly sample efficient, and their well-calibrated uncertainty estimates make them effective surrogate models for Bayesian optimisation [ŌHagan, 1978] and active learning [Seo et al., 2000], as well as useful dynamics models for model-based reinforcement learning [Deisenroth and Rasmussen, 2011]. Inference becomes rapidly more expensive as the dataset grows however, motivating more computationally efficient approximations [Snelson and Ghahramani, 2005, Hensman et al., 2013, Titsias, 2009]. Of particular relevance to this thesis are tree-structured GP approximations [TSGP; Bui and Turner, 2014], which factorise a dense GP prior into blocks. This factorisation enables the distribution of both the model and the inference computation. We use this for efficient multi-robot learning in Chapter 5.

#### 1.2.2.3 Bayesian Deep Learning

Bayesian DL [MacKay, 1992, Neal, 1995] is the application of Bayesian inference to the weights of a NN, i.e. we have a NN with parameters $\theta$, and we aim to apply (1.19) to get the posterior over parameters $p(\theta|\mathcal{D})$. The "promise" of Bayesian DL is the benefit of expressive NN models, while enjoying the desirable properties of Bayesian modelling such as those listed in Section 1.2.2.1. In addition, many propose Bayesian model selection provides a more principled and efficient means to select DL hyperparameters like network architectures and prior/regularisation strength than cross-validation [MacKay, 1992, Immer et al., 2021].

Exact inference in NN models is intractable, and many methods of approximate inference for Bayesian DL have been proposed. Early works focused on the Laplace approximation [MacKay, 1992] and Hamiltonian Monte Carlo [HMC; Neal, 1995]. However, these were both working with small datasets and small networks by today's standards. More recent work aiming to scale Bayesian DL to large models and datasets has employed variational inference [Graves, 2011, Blundell et al., 2015, Gal and Ghahramani, 2016]; Markov chain Monte Carlo

(MCMC) methods such as stochastic gradient Langevin dynamics [Wenzel et al., 2020] or stochastic HMC [Chen et al., 2014]; efficient Kronecker factored Laplace approximations [Immer et al., 2021, Ritter et al., 2018] and expectation propagation [Hernández-Lobato and Adams, 2015]. Further, deep ensembles have also been argued to constitute Bayesian DL. They randomly initialise and independently train a collection of network replicas, and average their predictions at test time. As the averaging over predictions can be interpreted as a Bayesian model average (1.20), D'Angelo and Fortuin [2021] and Wilson and Izmailov [2020] argue they may be viewed as Bayesian neural networks (BNNs).

**Cold Posterior Effect** Despite theoretical promise, Bayesian DL is not widely used in practice. While the additional computation required for Bayesian inference (1.19) over loss minimisation (1.1) may provide a reason for this, the poor empirical performance of the Bayes posterior observed by Wenzel et al. [2020] is another possible explanation. In particular, they show that the Bayesian model average predictions can be less accurate than a point estimate of the weights. However, they find that in these settings, the *cold posterior*

$$p_T\left(\theta|\mathcal{D}\right) \propto p\left(\theta|\mathcal{D}\right)^{\frac{1}{T}}, \qquad T << 1 \tag{1.27}$$

is significantly more accurate than both the Bayes posterior ($T = 1$) and the point estimate, suggesting maintaining a distribution over weights is indeed beneficial. This cooling transformation concentrates the distribution around its modes, and leads to the larger modes becoming more dominant (Figure 1.3).

This observation is concerning, in part because the Bayes posterior *should* be optimal for the correct model [Jaynes, 2003], but also in practice, because it introduces an additional hyperparameter $T$ which must be tuned. Multiple explanations have been proposed for the cause of the cold posterior effect: incorrect likelihood for data with low aleatoric uncertainty [Adlam et al., 2020, Aitchison, 2020], incorrect prior distribution [Wenzel et al., 2020, Fortuin et al., 2022], the presence of DA during training [Wenzel et al., 2020, Izmailov et al., 2021], or a combination of all three [Noci et al., 2021]. In Chapter 3, we investigate the relationship between DA and the cold posterior effect, focusing on how DA impacts the likelihood function used during training.

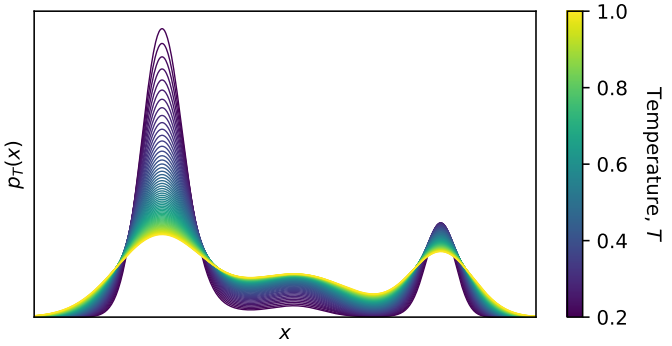



Figure 1.3: **The effect of cooling on probability density.** The temperature transformation (1.27) for $T < 1$ causes the distribution to become more concentrated around the modes, and for the larger modes to dominate.

### 1.2.3 Bayes and Computation

What does the "Bayesian recipe" (1, 2 and 3 in Section 1.2.1) imply about computation? At first glance, it seems to say little about exactly *how* to compute the quantities of interest, but let us consider an example for further exploration.

#### 1.2.3.1 An Example Inference Problem

Suppose we have a generative model which we write in an undirected form

$$p(a,b,c,d,e,f) = \frac{1}{Z}\phi_1(a)\,\phi_2(a,b)\,\phi_3(c)\,\phi_4(c,d)\,\phi_5(b,d,e)\,\phi_6(e,f)\,\phi_7(f)\,. \tag{1.28}$$

The factor graph is presented in Figure 1.4a. Now, let's say we observe $a = \alpha$, $c = \gamma$, and want to estimate $e$, i.e. we seek $p(e|a = \alpha, c = \gamma)$. We will follow the procedure outlined in Section 1.2.1: condition on observations and marginalise out nuisance variables $b, d, f$.

The conditioned target is

$$p(e|a = \alpha, c = \gamma) = \frac{1}{Z'}\sum_{b,d,f}\phi_2(a = \alpha, b)\,\phi_4(c = \gamma, d)\,\phi_5(b,d,e)\,\phi_6(e,f)\,\phi_7(f) \tag{1.29}$$

where we have subsumed the additional terms which do not depend on $f$ into a new $Z'$. Now let us assume we want to calculate (1.29) as efficiently as possible. To do so, we must rearrange the summations so they are executed over the minimal number of factors. This reduces the number of the combinations of variables we have to sum over. Rearranging gives

$$p\left(e|a=\alpha, c=\gamma\right) = \frac{1}{Z'} \sum_{b} \left( \phi_2\left(a=\alpha, b\right) \sum_{d} \left( \phi_4\left(c=\gamma, d\right) \phi_5\left(b, d, e\right) \right) \right) \sum_{f} \left( \phi_6\left(e, f\right) \phi_7\left(f\right) \right) \tag{1.30}$$

This expression can be broken down into five stages:

$$m_{2b} \leftarrow \phi_2\left(a=\alpha, b\right) \tag{1.31}$$

$$m_{4d} \leftarrow \phi_4\left(c=\gamma, d\right) \tag{1.32}$$

$$m_{5e} \leftarrow \sum_{b,d} m_{2b} m_{4d} \phi_5\left(b, d, e\right) \tag{1.33}$$

$$m_{7f} \leftarrow \phi_7\left(f\right) \tag{1.34}$$

$$m_{6e} \leftarrow \sum_{f} m_{7f} \phi_6\left(e, f\right) \tag{1.35}$$

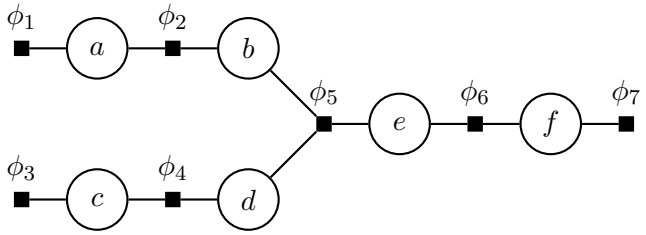
and the final posterior can be computed as

$$p\left(e|a=\alpha, c=\gamma\right) = \frac{1}{Z'} m_{5e} m_{6e}. \tag{1.36}$$

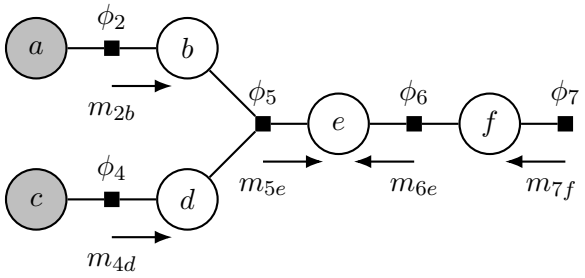
Examining the conditioned factor graph (Figure 1.4b), we see that these intermediate quantities are local signals – *messages* – which are passed along the graph in order to compute the posterior $p\left(e|a=\alpha, c=\gamma\right)$.

To summarise, we i) defined a generative model, ii) conditioned it on some observations, and iii) marginalised out the nuisance variables. Simply by rearranging the computation to be as efficient as possible, we found this "Bayesian recipe" gave rise to an algorithm which uses entirely distributed computation: messages are calculated based on the local state of the model ($\{m_{ij}\}$ and factors $\{\phi_k\}$) and passed around to compute the target posterior.

The algorithm is called belief propagation, and was introduced by Pearl [1982]. It is in fact a general method for marginal inference of a joint distribution and is exact in tree-structured models. It has also been widely successful in models containing cycles [see e.g. Gallager, 1962, Murphy et al., 1999, George et al., 2017, Lázaro-Gredilla et al., 2016].




(a) Without observation.




(b) With observed $a$ and $c$.

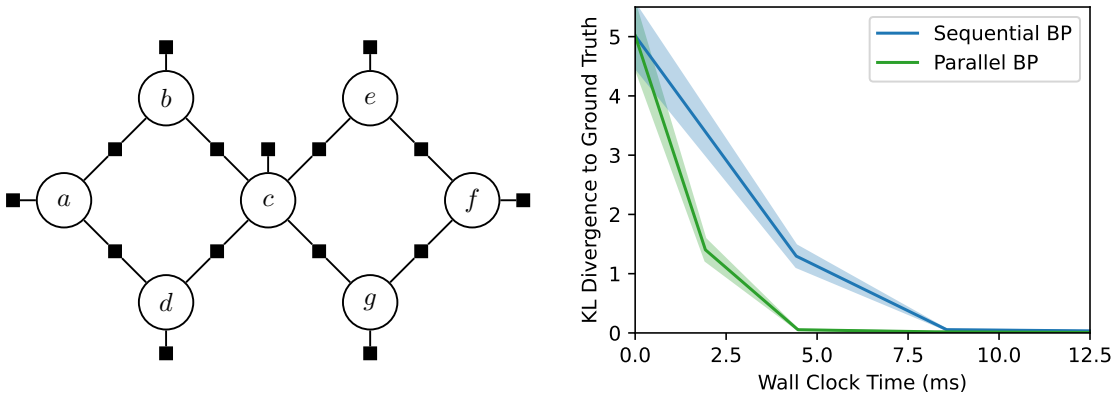
Figure 1.4: **The factor graph for model** (1.28). Arrows represent messages which are passed during inference of $e$.

**Distributed *and* Parallel?** While we have shown that Bayesian inference can be carried out with distributed computation and message passing, one might question the benefit of this in a model such as (1.28) (Figure 1.4a). Despite being able to compute $m_{6e}$ and $m_{5e}$ in parallel, the message passing routine in (1.31) to (1.35) has many sequential dependencies and therefore may suffer from the same "locking" which prevents model parallelism with the backpropagation algorithm (Section 1.1.1.3, point 3).

Our example model is a tree which means the most efficient schedule is to "sweep" through the model in order, waiting until all preceding messages are calculated before computing the outgoing at each stage. In this case, unsynchronised computation will be wasted as it will have to be recomputed after each dependency has been updated.

For loopy models, however, the optimal schedule is not clear as an updated message will cycle around the graph and affect the incoming messages on which it depends. In this case, inference is approximate and iterative – one typically repeats a chosen schedule of message updates until convergence. Parallelism (enabled by the locality of the update rules) *is* then beneficial as it enables different regions of the graph to reach local agreement more quickly.

We demonstrate this speedup with a toy example of a small Gaussian model (Figure 1.5).




(a) Loopy factor graph example (b) Wall clock time convergence

Figure 1.5: **Toy demonstration of the parallel speedup of loopy BP.** We run BP in the factor graph (a), and measure the accuracy of the inference on the marginal of node $c$ (b). A parallel message schedule which simultaneously updates the messages within the two loops converges more quickly than the serial implementation (b). The shaded region covers one standard error either side of the mean, over 100 models with randomly sampled factors.

We do BP inference of central node's marginal $p(c)$, using two message update schedules: i) a sequential schedule in which all messages are updated in serial, and ii) a parallel schedule in which we update the messages in the two loops simultaneously. The parallel schedule converges at nearly double the speed (Figure 1.5b).

In general, the BP message schedule is a design choice set by the practitioner, but a large body of empirical evidence demonstrates that asynchronous message schedules usually converge [Elidan et al., 2006, Wang et al., 2005, Ortiz et al., 2021, Ranganathan et al., 2007, Murai et al., 2023, Patwardhan et al., 2022], and do so more often than synchronous updates. This is a highly desirable property as it implies global coordination of message updates is not necessary, further supporting the case for parallel, decentralised inference with BP.

#### 1.2.3.2 Gaussian Belief Propagation

While BP has traditionally been applied to discrete models, the algorithm is agnostic to the form of the factors. BP in models with Gaussian factors, which we refer to as Gaussian BP [GBP; Bickson, 2008, Ortiz et al., 2021], has been applied as a distributed inference engine in a broad variety of continuous systems. A non-exhaustive list of recent applications includes multi-robot planning [Patwardhan et al., 2022], collaborative localisation [Murai

et al., 2023], simultaneous localisation and mapping [SLAM; Ranganathan et al., 2007, Hug et al., 2024, Ortiz et al., 2022], gas distribution inference [Rhodes et al., 2023], rotational odometry [Alzugaray et al., 2024], automated calibration [Murai et al., 2024], optical flow [Nagata and Sekikawa, 2023] and fluid dynamics modelling [MacKinlay et al., 2025].

Recognising that Gaussian factors are often a poor approximation, multiple methods have been proposed to generalise GBP to accommodate non-Gaussian, non-linear factors. Davison and Ortiz [2019] present an iterative linearisation scheme which locally approximates a non-linear factor as Gaussian, runs GBP and updates the linearisation point based on the current marginal estimates. Minka [2013] instead approximate the outgoing message as Gaussian via moment matching. Both of these approaches make GBP applicable to a broader class of problems, enabling approximate inference in more accurate models.

The properties of GBP are largely the same as BP in general models: it is a natively distributed algorithm in which message updates are computed in-place and depend only on the local state of the graph. Moreover, it is also guaranteed to converge in tree-structured models, but not in models with cycles. However, *if* GBP converges in loopy graphs, the posterior means are guaranteed to be exact [Weiss and Freeman, 1999], which is not true for BP in arbitrary loopy models.

In Chapter 4, we employ GBP as an inference engine for distributed learning and prediction in deep models. We go on to show, in Chapter 5, how GBP can also enable the distribution of learning in multi-robot settings, where each robot estimates a local function from their own experience and inter-robot messages ensure neighbouring robots' estimates are consistent.

**A Changing Hardware Landscape.** We noted that part of DL's success was a well-matched and highly optimised hardware platform in the form of GPUs and similar processors (point 1, Section 1.1.1.2). This is a prerequisite to any successful algorithm according to both Sutton's bitter lesson [Sutton, 2019] and Hooker's hardware lottery [Hooker, 2021]. It is therefore prudent to assess suitable hardware options for running distributed computation like BP.

We highlight two processors which are particularly well-suited: Graphcore's intelligence processing unit [IPU; Jia et al., 2019] and the Cerebras wafer scale engine [WSE; Cerebras]. Both operate on a multiple instruction multiple data (MIMD) paradigm, where each core to be running a different program in parallel (in contrast to a GPU which executes SIMD parallelism). Further, both have processing cores with significant local memory. These

features combined imply an arbitrary graph can be mapped onto the processor and different regions can be processed in parallel without frequent exchanges. Indeed, Ortiz et al. [2020] run a belief propagation solver for bundle adjustment on an IPU and achieve a 24× speedup over an efficient CPU implementation.

In addition to graph processors, we see a growing interest in heterogeneous, decentralised hardware clusters, which could naturally accommodate graph-based computation [Tarafdar et al., 2020, Douillard et al., 2023, Xu et al., 2023, Mei et al., 2025, Exolabs, 2025].

## 1.3 Contributions

This thesis covers work presented in the following publications:

- Seth Nabarro, Stoil Ganev, Adrià Garriga-Alonso, Vincent Fortuin, Mark van der Wilk, and Laurence Aitchison. Data augmentation in Bayesian neural networks and the cold posterior effect. In *Uncertainty in Artificial Intelligence*, pages 1434–1444. PMLR, 2022 [Nabarro et al., 2022]. First authorship shared with Stoil Ganev.
- Seth Nabarro, Mark van der Wilk, and Andrew J Davison. Learning in deep factor graphs with Gaussian belief propagation. In *International Conference on Machine Learning*, pages 37141–37163. PMLR, 2024 [Nabarro et al., 2024].
- Seth Nabarro, Mark van der Wilk, and Andrew J Davison. A distributed Gaussian process model for multi-robot mapping. Under review at *2026 International Conference on Robotics and Automation (ICRA)*, 2025 [Nabarro et al., 2025].

## 1.4 Thesis Structure

We structure the remaining chapters as follows:

- **Chapter 2: Preliminaries.** The next chapter presents the necessary background for the remainder of the thesis. Here we introduce probability theory and probabilistic models, then discuss different types of inference. We finish by reviewing the theory and practice of BP.

- **Chapter 3: Data Augmentation in Bayesian Neural Networks and the Cold Posterior Effect.** Here we discuss how DA should be incorporated into Bayesian inference. We present two novel bounds on the categorical likelihood of an invariant predictor, and evaluate their impact on the cold posterior effect in Bayesian DL. Last, we contrast the behaviours of the two bounds, showing they induce posteriors with different characteristics. [Nabarro et al., 2022].

- **Chapter 4: Learning in Deep Factor Graphs with Gaussian Belief Propagation.** In this chapter we introduce *GBP Learning*, a method for distributed learning in deep models with GBP. We detail the design of NN-inspired multi-layer factor graphs, and discuss how GBP inference in such models can be made efficient. Empirically, we show encouraging performance for continual learning, image denoising and image classification; and demonstrate how learnt and hand-designed components can be integrated into a single model [Nabarro et al., 2024].

- **Chapter 5: A Distributed Gaussian Process Model for Multi-Robot Mapping.** Next, we propose a method for multiple robots to learn a global function with only local experience, computation and communication. Each robot maintains a local model and GBP messages between nearby robots ensure their representations are consistent. We demonstrate superior performance against existing distributed learning techniques [Nabarro et al., 2025].

- **Chapter 6: Conclusion.** We finish with a summary of the thesis and some closing thoughts as to future work.

# Chapter 2

# Preliminaries

In this chapter we review the background relevant to the remainder of the thesis. We will begin with a brief discussion as to the meaning of probability, then present an overview of probabilistic graphical models, before looking at sampling-based approximate inference techniques. For belief propagation (BP), we first derive the algorithm for tree-structured models, then discuss BP in models containing cycles, and details of the algorithm for Gaussian models. We finish with a short introduction to Gaussian processes.

## 2.1 What is Probability?

There are two distinct perspectives on the meaning of probability. In the *frequentist* view [Neyman, 1977], probabilities are rates: the fraction of the time that an event occurs, on average, in the limit of infinite samples. In contrast, the *Bayesian* view [Jaynes, 2003, MacKay, 2003] says that probabilities represent degrees of belief, which depend on the assumptions of the modeller and their state of knowledge. Given assumptions can be chosen arbitrarily, the Bayesian perspective is also known as the *subjective* view.

Can we reasonably represent beliefs with probabilities? The relation between belief and probability is made concrete by Cox's theorem [Cox, 1946], which states that if beliefs are *consistent* in the sense of Cox's axioms:

1. **Beliefs can be ordered.**, If we say $B(a) > B(b)$, and $B(b) > B(c)$, then it follows that $B(a) > B(c)$. If beliefs can be ordered in this way, then they can be mapped onto real numbers.

2. **Beliefs in a statement $a$ and its negation $\bar{a}$ are related**, i.e. there is a function $g(\cdot)$ such that $B(a) = g\left(B\left(\bar{a}\right)\right)$.

3. **Joint beliefs $B(a, b)$ relate to conditional $B\left(a|b\right)$ and marginal $B\left(b\right)$ beliefs,** i.e. there is a function $h(\cdot)$ such that $B(a, b) = h\left(B\left(a|b\right), B\left(a\right)\right)$.

then they are equivalent to probabilities and therefore follow the rules of probability. This justifies the use of probability theory as the rational approach to updating beliefs.

While the subjectivity of the Bayesian view may seem alarming, the assumptions as to how the data were generated allow us to incorporate prior knowledge, eliminate the effect of nuisance variables, and deduce our updated belief about others [Jaynes, 2003, MacKay, 2003]. This is not possible in a frequentist framework. As MacKay [2003] emphatically states: *you cannot do inference without making assumptions.*

### 2.1.1 Rules of Probability

We denote the joint probability over $\mathbf{a}$ and $\mathbf{b}$ as $p\left(\mathbf{a}, \mathbf{b}\right)$. Any valid joint distribution satisfies the product rule

$$p\left(\mathbf{a}, \mathbf{b}\right) = p\left(\mathbf{a}|\mathbf{b}\right) p\left(\mathbf{b}\right), \tag{2.1}$$

and the sum rule

$$p\left(\mathbf{a}\right) = \sum_{\mathbf{b}} p\left(\mathbf{a}, \mathbf{b}\right), \tag{2.2}$$

where $p\left(\mathbf{a}\right)$ is known as the marginal distribution of $\mathbf{a}$.

## 2.2 Probabilistic Graphical Models

We have stated that Bayesian modelling is based on subjective assumptions. These assumptions are expressed as a *model* – a hypothesis about how the observations are generated in the form of a joint probability distribution. This can be specified as a graph, known as a probabilistic graphical model (PGM), which intuitively describes how the variables relate to one another. There a many classes of PGM, some for which the direction of the connections matter (directed PGMs – DGMs) and others for which they don't (undirected PGMs – UGMs). We present an example of a directed PGM below.

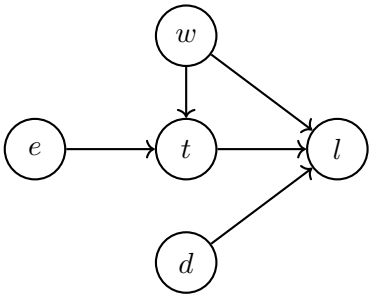



Figure 2.1: **The directed PGM for an example model of bus timing.** We assume the bus arrival time $l$ depends on weather $w$, traffic $t$, driver availability $d$ and large events in the area $e$. See text for details.

Consider a simple model of bus timing. We denote the event of a bus arriving late to a stop as $l$, and we assume that this depends on e.g. $t$ current traffic, $w$ current weather, $d$ whether there are a sufficient number of drivers (e.g. due to illness/strikes/holiday), $e$ whether there is a popular event (e.g. sports) in the area. One might make the following assumptions which dictate the PGM structure:

1. Weather $w$ affects both the traffic $t$ and the bus timing $l$ directly as all vehicles may drive more slowly in rain, causing more traffic, but also a slower bus in the absence of traffic.

   $\Rightarrow$ The PGM has edges $w \to t$ and $w \to l$.

2. The probability of a popular event $e$ is independent of weather $w$ and number of drivers $d$. Similarly $w$ and $d$ are independent.

   $\Rightarrow$ There are no edges between $d$, $e$ and $w$.

3. An popular event $e$ only affects the bus timing $l$ due to the additional traffic it causes $t$.

   $\Rightarrow$ There is an edge $e \to t$, but no direct edge between $e$ and $l$.

We draw the corresponding PGM in Figure 2.1. While the purpose of this model may currently be unclear, we will later discuss inference (Section 2.3) which enables one to "ask questions of the model". For example: *given my bus arrived* 5 *minutes late, and it is raining, what is the probability that there is an event today?*

#### 2.2.0.1 Marginal and Conditional Independence

The structure of probabilistic models encode independence and conditional independence relationships. Variables $a$ and $b$ can be said to be (marginally) independent if their joint distribution factorises into the product of marginals

$$a \perp b \iff p(a,b) = p(a)\,p(b)\,. \tag{2.3}$$

which Murphy [2012] notes is rare, as it is common for variables in a model to interact. More often one encounters *conditional* independence. We say $a$ and $b$ are conditionally independent given $c$ if their conditional joint decomposes

$$a \perp b \mid c \iff p(a,b|c) = p(a|c)\,(b|c)\,. \tag{2.4}$$

PGMs neatly describe a model's independence structure. Intuitively, models with fewer edges have more conditional independence relationships than highly connected models. Independence in a DGM can be deduced via the rules of *d-seperation* [Murphy, 2012]. Specifically, two variables, $a$ and $b$ are independent given the conditioning set of nodes $\mathcal{V}$ iff every path between $a$ and $b$ is d-separated. A path is d-separated by $\mathcal{V}$ iff

- contains a "chain" $a \to m \to b$, or $a \leftarrow m \leftarrow b$ where $m \in \mathcal{V}$,
- contains a "fork" $a \leftarrow m \to b$ where $m \in \mathcal{V}$, or
- contains a "collider" $a \to m \leftarrow b$ where $m \notin \mathcal{V}$.

Note that if $\mathcal{V} = \emptyset$, d-separation is a test for marginal, rather than conditional, independence.

In UGMs, conditional independence is more straightforward: $a$ and $b$ are independent given $c$, iff every path between $a$ and $b$ goes through $c$. Thus if no direct edge exists between $a$ and $b$, then they are independent given all the other nodes in the graph.

In general, UGMs are unable to represent some of the conditional independences of DGMs, and vice versa. For example, for a collider $a \to c \leftarrow b$ in a directed model, we have $a \perp b$ and $a \not\perp b, \mid c$, but the closest undirected model $a - b - c$ which would mean $a \not\perp b$ and $a \perp b \mid c$. On the other hand, an UGM comprising four variables connected in a ring, cannot be accurately translated to a DGM. Attempts to do so create a collider node which results in a different independence structure from the original undirected model. See [Murphy, 2012, Section 19.2.3] for further discussion.

### 2.2.1 Factor Graphs

While UGMs such as Markov random fields (MRF) have more straightforward conditional independence properties than DGMs, they suffer from representational ambiguity: multiple factorisations of the joint distribution may result in the same MRF. Factor graphs, in contrast, have a one-to-one relationship between the graph and the joint distribution.

The Hammersley-Clifford theorem [Hammersley and Clifford, 1971] says that any UGM with a positive probability function $p(\mathbf{x})$ can be represented as a factor graph, that is

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c) \tag{2.5}$$

where $\mathcal{C}$ are the maximal cliques[1] of the undirected graph, $\{\phi_c(\mathbf{x}_c)\}_{c \in \mathcal{C}}$ are the positively valued factor or potential functions, and $Z := \sum_{\mathbf{x}} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c)$ is a normalising constant also known as the partition function. We use $\mathbf{x}_c$ to denote the variables in maximal clique $c$, which could also be written as $\{x_j\}_{j \in \text{nei}(\phi_c)}$, were nei$(\cdot)$ returns the neighbours of the node passed as input. The factorisation (2.5) can be viewed as a graph in which factor nodes connect to the variable nodes on which they depend, hence the name *factor graph*. An example factor graph is given in Figure 1.4.

As with other PGMs, factor graphs make the structure of the model clear and expose potential computational efficiencies. They share the same, straightforward conditional independence properties as MRFs – variables $a$ and $b$ are conditionally independent given $c$ if all paths between $a$ and $b$ go through $c$. However, they are a precise representation of the joint factorisation, where MRFs are ambiguous. Further, compared to DGMs which comprise normalised conditional distributions, factor graphs are better suited to inference as they do not require the estimation of expensive normalisation factors [Dellaert et al., 2017]. We will see that the BP algorithm (Section 2.4) avoids computation of $Z$ and leverages computational efficiencies arising from model structure.

**Relation to Energy-Based Models.** Factor graphs are closely related to energy-based models [LeCun et al., 2006, Du and Mordatch, 2019] which are defined by the Gibbs

[1] A maximal clique $\mathcal{H}$ of graph $\mathcal{G}$ is a subgraph of $\mathcal{G}$ which is i) a clique – all nodes in $\mathcal{H}$ connect to all others, and ii) maximal – there are no nodes in $\mathcal{G}$ which are not in $\mathcal{H}$ but connect to all nodes in $\mathcal{H}$.

distribution

$$p(\mathbf{x}) = \frac{1}{Z} \exp\left(-\sum_{c \in \mathcal{C}} E(\mathbf{x}_c)\right) \tag{2.6}$$

where $E(\cdot)$ is known as the *energy function*. Comparing (2.5) and (2.6) we can see that the models are equivalent with $\phi_c(\mathbf{x}_c) = \exp(-E(\mathbf{x}_c))$.

## 2.3 Probabilistic Inference

From the product (2.1) and sum rules (2.2) of probability we can derive Bayes rule

$$p(\mathbf{a}|\mathbf{b}) = \frac{p(\mathbf{a}, \mathbf{b})}{p(\mathbf{b})} \tag{2.7}$$

$$= \frac{p(\mathbf{b}|\mathbf{a})\, p(\mathbf{a})}{\int_{\mathbf{a}} p(\mathbf{b}|\mathbf{a})\, p(\mathbf{a})\, \mathrm{d}\mathbf{a}}, \tag{2.8}$$

the basis of probabilistic inference. It relates a chosen prior distribution $p(\mathbf{a})$ and likelihood $p(\mathbf{b}|\mathbf{a})$ (assuming we observe $\mathbf{b}$) to an updated belief of $\mathbf{a}$, $p(\mathbf{a}|\mathbf{b})$.

In practice, $p(\mathbf{a}|\mathbf{b})$ is usually intractable, due to the numerator $p(\mathbf{b}) = \int_{\mathbf{a}} p(\mathbf{b}|\mathbf{a})\, p(\mathbf{a})\, \mathrm{d}\mathbf{a}$ either having no closed-form solution or being prohibitively expensive to compute. In these cases, one must resort to either *approximate inference* techniques, such as Markov chain Monte Carlo (MCMC), or reducing the scope of inference instead of the full posterior distribution. We briefly define the different types of inference:

- **Posterior inference.** In this case the goal is to compute or approximate the full posterior distribution $p(\mathbf{a}|\mathbf{b})$.
- **Maximum a-posteriori (MAP) inference.** Here we seek to find the mode of the posterior $\mathbf{a}^* = \operatorname{argmax}_{\mathbf{a}} p(\mathbf{a}|\mathbf{b})$. A common method for MAP inference is to do gradient ascent of $p(\mathbf{a}, \mathbf{b})$ with respect to $\mathbf{a}$, as $\operatorname{argmax}_{\mathbf{a}} p(\mathbf{a}, \mathbf{b}) = \operatorname{argmax}_{\mathbf{a}} p(\mathbf{a}|\mathbf{b})$.
- **Marginal inference.** Suppose $\mathbf{a} = [a_1, a_2, \ldots, a_{D_{\mathbf{a}}}]$, where $D_{\mathbf{a}} := \dim(\mathbf{a})$. The full posterior $p(\mathbf{a}|\mathbf{b})$ may be a complicated and computationally expensive object in cases where we are only interested in a small subset of the elements of $\mathbf{a}$. If so, *marginal inference* of e.g. $p(a_i|\mathbf{b})$ may be more appropriate.

### 2.3.1 Markov Chain Monte Carlo

In most cases, probabilistic inference is not an end in itself but a means to achieve a downstream objective. For example, we may do inference on this afternoon's weather in order to make a decision about what we will wear, or do inference on the pose of a robot in order to determine the appropriate control signals to reach a goal location. We are almost always making a *decision*, $\mathbf{d}$ based on the results of our inference $p(\mathbf{a}|\mathbf{b})$. The optimal decision can be formalised as the optimisation

$$\mathbf{d}^* = \text{argmin}_{\mathbf{d}} \mathbb{E}_{p(\mathbf{a}|\mathbf{b})} \left[ l(\mathbf{a}, \mathbf{d}) \right] \tag{2.9}$$

where $l(\mathbf{a}, \mathbf{d})$ is the penalty of deciding $\mathbf{d}$ when the true state is $\mathbf{a}$. The expectation of $l(\mathbf{a}, \mathbf{d})$ wrt $p(\mathbf{a}|\mathbf{b})$ is known as *Bayes risk*.

As the Bayes risk is an expectation over our posterior distribution, Monte Carlo integration can provide an unbiased estimator

$$\mathbf{d}^* \approx \text{argmin}_{\mathbf{d}} \frac{1}{N_a} \sum_{i=1}^{N_a} \left[ l(\mathbf{a}_i, \mathbf{d}) \right], \quad \mathbf{a}_i \sim p(\mathbf{a}|\mathbf{b}), \ i = 1, \ldots, N_a. \tag{2.10}$$

For settings where full posterior inference is intractable, it may therefore suffice to have representative samples $\{\mathbf{a}_i\}_{i=1}^{N_a}$ which we can use for downstream decision making.

MCMC [Gelman et al., 1995] is a set of techniques which seek to sample from non-trivial probability distributions. It is a commonly used tool for drawing posterior samples while bypassing the need to estimate the distribution itself. The main idea is simulate a Markov chain $\mathbf{a}_1, \ldots, \mathbf{a}_T$ whose stationary distribution is the posterior, thus as $T \to \infty$ these samples approach posterior samples. Chains are generated by repeatedly applying a one-step transition operator $F(\mathbf{a}_t|\mathbf{a}_{t-1})$ to perturb the current state. If particular characteristics of this operator can be proven, then we are guaranteed to draw posterior samples as $T \to \infty$. We will now introduce two transition operators: Metropolis-Hastings and Stochastic Gradient Langevin Dynamics (SGLD). We refer the reader to Gelman et al. [1995] for an in depth discussion of MCMC methods.

#### 2.3.1.1 Metropolis-Hastings

The Metropolis-Hastings algorithm is one of the most straightforward MCMC transition operators. It iteratively and randomly perturbs the current state, then accepts or rejects

the result. At each step we

1. perturb the state by sampling from a chosen "jump" distribution $\mathbf{a}'_t \sim J\left(\mathbf{a}_t|\mathbf{a}_{t-1}\right)$,
2. compute the acceptance probability,

$$r = \frac{p\left(\mathbf{a}'_t, \mathbf{b}\right)/J\left(\mathbf{a}'_t|\mathbf{a}_{t-1}\right)}{p\left(\mathbf{a}_{t-1}, \mathbf{b}\right)/J\left(\mathbf{a}_{t-1}|\mathbf{a}'_t\right)} \tag{2.11}$$

3. accept the proposed state $\mathbf{a}'_t$ with probability $r$, else reject it and stay at $\mathbf{a}_{t-1}$

$$u_t \sim \text{Unif}(0,1) \tag{2.12}$$

$$\mathbf{a}_t = \begin{cases} \mathbf{a}'_t, & \text{if} \quad u_t < \min(r,1) \\ \mathbf{a}_{t-1} & \text{otherwise} \end{cases}. \tag{2.13}$$

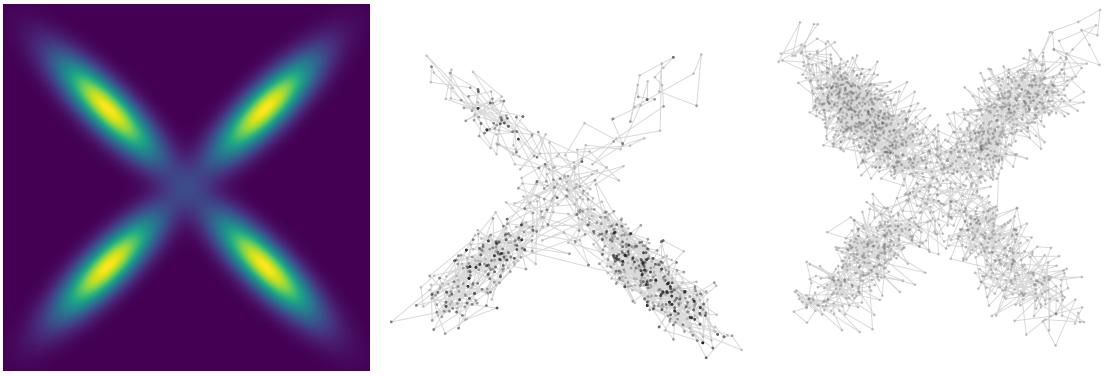
Note that we do not need to evaluate the posterior density function $p\left(\mathbf{a}'_t|\mathbf{b}\right)$ in (2.11), only the joint distribution $p\left(\mathbf{a}'_t, \mathbf{b}\right)$ which is proportional to the posterior in its dependence on $\mathbf{a}$. The constants of proportionality cancel.

While Metropolis-Hastings can be proven to be a valid MCMC kernel, it is often inefficient. $J\left(\mathbf{a}'_t|\mathbf{a}_{t-1}\right)$ is typically chosen to be a straightforward axis-aligned Gaussian, which means a uninformed perturbation is applied which can result in proposals far away from regions of high density. The perturbed state is therefore likely to have a low acceptance probability $r$, particularly if the size of the perturbation is large.

#### 2.3.1.2 Stochastic Gradient Langevin Dynamics

SGLD [Welling and Teh, 2011] is an MCMC technique combining stochastic (minibatch), gradient-based optimisation [Robbins and Monro, 1951] with noise-injecting Langevin Monte Carlo [Neal et al., 2011]. The two are complimentary: without the noise from the Langevin sampling, the optimisation would converge to a MAP solution, however the minibatch optimisation techniques allow posterior sampling to be scaled to large datasets.

Suppose we are targeting a posterior over parameters given observations with independent likelihoods $p\left(\theta|\mathcal{D}\right) \propto p\left(\theta, \mathcal{D}\right) = p\left(\theta\right)\prod_{i=1}^{N} p\left(\mathcal{D}_i|\theta\right)$. Stochastic gradient ascent of the log-



(a) Target distribution (b) Metropolis-Hastings (c) Langevin

Figure 2.2: **Comparison between Metropolis-Hastings and Langevin MCMC.** We run Langevin (c) and Metropolis-Hastings (b) for $2 \times 10^3$ steps targeting the distribution presented in (a). Langevin uses the gradient of the target and so is better able to cover the distribution than Metropolis-Hastings, which takes random, uninformed steps.

posterior follows the updates

$$\theta_t = \theta_{t-1} + \frac{\alpha_{t-1}}{2} \nabla_\theta \left( \log p\left(\theta_{t-1}\right) + \frac{N}{n} \sum_{j=1}^{n} \log p\left( \mathcal{D}_j^{(t-1)} \middle| \theta_{t-1} \right) \right) \tag{2.14}$$

where $\mathcal{D}^{(t-1)}$ is the sampled minibatch of $n$ examples at step $t-1$. $\alpha_0, \alpha_1, \ldots$ is a sequence of stepsizes whose schedule must satisfy

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \sum_{t=1}^{\infty} \alpha_t < \infty \tag{2.15}$$

to ensure convergence to a local maximum [Robbins and Monro, 1951].

SGLD augments stochastic optimisation with additive Gaussian noise

$$\theta_t = \theta_{t-1} + \frac{\alpha_{t-1}}{2} \nabla_\theta \left( \log p\left(\theta_{t-1}\right) + \frac{N}{n} \sum_{j=1}^{n} \log p\left( \mathcal{D}_j^{(t-1)} \middle| \theta_{t-1} \right) \right) + \epsilon_{t-1} \tag{2.16}$$

$$\epsilon_{t-1} \sim \mathcal{N}\left(0, \alpha_{t-1}\right) \tag{2.17}$$

where $\alpha_0, \alpha_1, \ldots$ must also satisfy (2.15).

In the early stages of SGLD sampling, the gradient term in (2.16) dominates as $\theta$ is far

from an optimum. After many steps, however, the samples approach an optimum, meaning the gradient term approaches zero and the noise term $\epsilon_t$ dominates, driving the chain to explore around the mode. Thus, the algorithm begins like stochastic gradient ascent, but tends to Langevin dynamics as $t \to \infty$.

SGLD is significantly more efficient than Metropolis-Hastings due to the gradient term providing better perturbation direction in low density regions. This enables SGLD to scale to high-dimensional problems (such as neural network posteriors [Wenzel et al., 2020]), where randomly perturbed Metropolis-Hastings would fail. We demonstrate this with a toy example (Figure 2.2) which shows Langevin dynamics provides better coverage of the distribution for the same number of samples.

## 2.4 Belief Propagation

BP [Pearl, 1982] is a method for doing marginal inference using message passing in a PGM. Messages are probability distributions which can be thought of as estimates of a variable's marginal distribution, made with only a partial view of the graph. At convergence, the aggregation of messages sent to a variable node provides an estimate of its full marginal distribution. We will now derive the BP algorithm for tree-structured models, following Bishop [2006].

### 2.4.1 Derivation for Tree-structured Models

Consider a tree-structured model,

$$p_{\text{tree}}(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c) \tag{2.18}$$

in which we aim to do marginal inference on node $x_i$. Any tree-structured factor graph can be divided into $N_{st} = |\text{nei}(x_i)|$ subtrees, with one extending from each branch which connects $x_i$ to a factor (see e.g. different shaded regions in Figure 2.3). Each subtree, indexed by $k$, will depend on both $x_i$, and some other variables $X_{k\setminus i}$

$$p_{\text{tree}}(\mathbf{x}) = \prod_{k=1}^{N_{st}} p_k\left(x_i, X_{k\setminus i}\right). \tag{2.19}$$

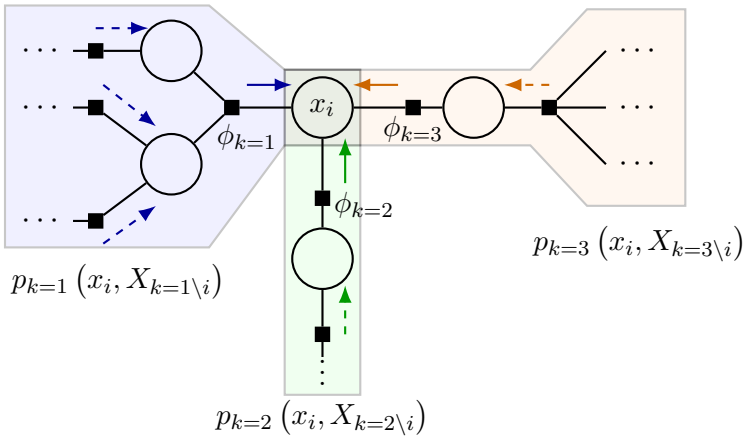



Figure 2.3: **An example of BP inference of $p(x_i)$ in a tree-structured model.** The model divides into $|\text{nei}(x_i)| = 3$ subtrees, each containing $x_i$ and disjoint sets of variables. Shaded regions of different colours cover different subtrees. Solid arrows represent messages from factors in the immediate neighbourhood $\text{nei}(x_i)$, (denoted $\{m_{\phi_k \to x_i}(x_i)\}_k$ in Section 2.4.1) and dashed arrows represent messages from factors one-step removed to the variables in $\text{nei}(\phi_k) \setminus x_i$ (denoted $\{\{m_{\phi_r \to x_l}(x_l)\}_{\phi_r \in \text{nei}(x_l)\setminus\phi_k}\}_{x_l \in \text{nei}(\phi_k)\setminus x_i}$).

Our inference target of marginal $p(x_i)$ is therefore

$$p(x_i) = \sum_{X_{\setminus i}} \prod_{k=1}^{N_{st}} p_k\left(x_i, X_{k\setminus i}\right) \tag{2.20}$$

$$= \prod_{k=1}^{N_{st}} \sum_{X_{k\setminus i}} p_k\left(x_i, X_{k\setminus i}\right), \tag{2.21}$$

where we have used the fact that the sum and product commute in the second equality. We define

$$m_{\phi_k \to x_i}(x_i) := \sum_{X_{k\setminus i}} p_k\left(x_i, X_{k\setminus i}\right). \tag{2.22}$$

We can decompose each subtree into the factor connected to $x_i$ and the remaining factors

$$p_k(\mathbf{x}) = \frac{1}{Z_k}\phi_k\left(x_i, \mathbf{x}_{k\setminus i}\right) \prod_{j \in \mathcal{C}_k \setminus \text{nei}(x_i)} \phi_j\left(\mathbf{x}_j\right), \tag{2.23}$$

where $\mathbf{x}_{k\setminus i}$ are the variables connected to factor $\phi_k$ except $x_i$, and $\mathcal{C}_k \setminus \text{nei}(x_i)$ are the indices of factors in subtree $k$ except that connected to $x_i$ ($\phi_k$). Substituting this expression into the definition of $m_{\phi_k \to x_i}(x_i)$ (2.22)

$$m_{\phi_k \to x_i}(x_i) = \frac{1}{Z_k} \sum_{X_{k\setminus i}} \phi_k\left(x_i, \mathbf{x}_{k\setminus i}\right) \prod_{j \in \mathcal{C}_k \setminus \text{nei}(x_i)} \phi_j\left(\mathbf{x}_j\right). \tag{2.24}$$

We can rewrite the product over remaining factors as the product over further subtrees

$$m_{\phi_k \to x_i}(x_i) = \frac{1}{Z_k} \sum_{X_{k\setminus i}} \phi_k\left(x_i, \mathbf{x}_{k\setminus i}\right) \prod_{x_l \in \text{nei}(\phi_k)\setminus x_i} \prod_{\phi_r \in \text{nei}(x_l)\setminus \phi_k} p_{r\setminus k}\left(x_l, X_{r\setminus l}\right) \tag{2.25}$$

$$= \frac{1}{Z_k} \sum_{\mathbf{x}_{k\setminus i}} \phi_k\left(x_i, \mathbf{x}_{k\setminus i}\right) \prod_{x_l \in \text{nei}(\phi_k)\setminus x_i} \prod_{\phi_r \in \text{nei}(x_l)\setminus \phi_k} \sum_{X_{r\setminus l}} p_{r\setminus k}\left(x_l, X_{r\setminus l}\right) \tag{2.26}$$

where $p_{r\setminus k}\left(x_l, X_{r\setminus l}\right)$ is the subtree branching from $\phi_r$ outwards (away from $\phi_k$ and $x_i$) and we have decomposed the summation over all variables in subtree $k$ except $i$ ($X_{k\setminus i}$) into those which connect to the shared factor with $x_i$ ($\mathbf{x}_{k\setminus i}$) and the remainder ($X_{r\setminus l}$).

Recognising that each term inside the product in (2.26) is of the same form as the definition of $m_{\phi_k \to x_i}(x_i)$, i.e. the marginalisation of all but one variables from a subtree, we denote these $m_{\phi_r \to x_l}(x_l) := \sum_{X_{r\setminus l}} p_{r\setminus k}\left(x_l, X_{r\setminus l}\right)$, so we have the recursion

$$m_{\phi_k \to x_i}(x_i) = \frac{1}{Z_k} \sum_{\mathbf{x}_{k\setminus i}} \phi_k\left(x_i, \mathbf{x}_{k\setminus i}\right) \prod_{x_l \in \text{nei}(\phi_k)\setminus x_i} \prod_{\phi_r \in \text{nei}(x_l)\setminus \phi_k} m_{\phi_r \to x_l}(x_l). \tag{2.27}$$

To simplify this expression we also define

$$m_{x_l \to \phi_k}(x_l) := \prod_{\phi_r \in \text{nei}(x_l)\setminus \phi_k} m_{\phi_r \to x_l}(x_l) \tag{2.28}$$

giving the cleaner form

$$m_{\phi_k \to x_i}(x_i) = \frac{1}{Z_k} \sum_{\mathbf{x}_{k\setminus i}} \phi_k\left(x_i, \mathbf{x}_{k\setminus i}\right) \prod_{x_l \in \text{nei}(\phi_k)\setminus x_i} m_{x_l \to \phi_k}(x_l). \tag{2.29}$$

Let us take stock. We i) began at our target node $x_i$ and recognised that for an assumed tree-structured model the graph is split into multiple subtrees, one for each connected factor (2.21), ii) decomposed each subtree $p_k(\mathbf{x}_k)$ into the immediate factor $\phi_k$, and the product

of (sub)subtrees which fork off from the variables connected to $\phi_k$ except $x_i$ (2.25), iii) exchanged the order of marginalisation, such that each sum is computed over the minimal set of factors (2.26) and iv) recognised a recursion and simplified (2.29).

The final algorithm involves three types of computation:

1. **Variable-to-factor message updates,** $m_{x_i \to \phi_k}(x_i)$, (2.28). Here we accumulate all messages incoming to a variable $x_i$, except that from the target factor $\phi_k$. Their product defines the variable-to-factor message.

2. **Factor-to-variable message updates,** $m_{\phi_k \to x_i}(x_i)$, (2.29), collects all the messages coming into a factor $\phi_k$ except that from the recipient variable $x_i$. The product of the these messages, multiplied by the factor potential $\phi_k(\mathbf{x}_k)$ is then marginalised of all variables except the target.

3. **Marginal belief updates.** The estimated belief of a variable is given by the product of all its incoming messages,

$$p(x_i) = \prod_{k=1}^{N_{st}} m_{\phi_k \to x_i}(x_i). \tag{2.30}$$

We believe the BP algorithm is remarkable for two reasons. First, it leverages the independence structure of the model for efficient computation. The result is that all of the updates are inherently local, with dependencies on quantities only in the immediate neighbourhood of the updated message. This implies it can be easily distributed and potentially parallelised; and moreover does not require expensive data movement between storage and processing. Second, the derived updates are conceptually straightforward relying on local execution of the well understood rules of probability – the sum (2.2) and product (2.1) rules. For this reason, BP is also known as the *sum-product algorithm*.

#### 2.4.1.1 Exact Optimisations

We outline two optimisations which can accelerate BP without approximation. First, if a variable is connected to more than one factor, it is suboptimal to compute all outgoing messages $\{m_{x_i \to \phi_r}(x_i)\}_{\phi_r \in \text{nei}(x_i)}$ as per (2.28). Instead, one can first calculate its belief

(2.30), $p(x_i)$ and then divide by the message from each recipient factor

$$m_{x_i \to \phi_r}(x_i) = \frac{p(x_i)}{m_{\phi_r \to x_i}(x_i)}. \tag{2.31}$$

to compute the outgoing message. This trick reduces the complexity of computing all outgoing messages from a variable, from $\mathcal{O}\left(|\text{nei}(x_i)|^2\right)$ to $\mathcal{O}\left(|\text{nei}(x_i)|\right)$. The second optimisation is for factors with exponential family density, i.e. can be written as

$$\phi(\mathbf{x}|\nu) \propto h(\mathbf{x}) \exp\left(\nu^\intercal T(\mathbf{x})\right) \tag{2.32}$$

[Murphy, 2012] where $\nu$ are the *natural* factor parameters, $T(\mathbf{x})$ is the *sufficient statistic* and $h(\mathbf{x})$ is a scaling constant, often $h(\mathbf{x}) = 1$. If factors follow this natural parameterisation, so too will the messages. The benefit of this is that products of messages are simply sums of the corresponding natural parameters, for example

$$\prod_{\phi_k \in \text{nei}(x_i)} m_{\phi_k \to x_i}(x_i) \propto \exp\left(\left(\sum_k \nu_k\right)^\intercal T(\mathbf{x})\right), \tag{2.33}$$

this enables efficient computation of the belief update (2.30) and the product of incoming messages to a factor (2.29). For example, in binary networks a factor's natural parameters are the log-odds, so the parameter of the marginal belief of $x_i$ is estimated by

$$\nu_{x_i} = \sum_{\phi_k \in \text{nei}(x_i)} \nu_{\phi_k \to x_i} = \sum_{\phi_k \in \text{nei}(x_i)} \log \frac{p_{\phi_k \to x_i}}{1 - p_{\phi_k \to x_i}}. \tag{2.34}$$

### 2.4.2 Marginalisation or Inference?

As we have outlined, BP is a means to compute a marginal distribution $p(x_i)$ from a joint $p(x_1, x_2, \ldots, x_i, \ldots, x_N)$. It might initially be unclear how this relates to inference, i.e. how is it helpful for estimating $p(\mathbf{x}|\mathbf{y})$, for observations $\mathbf{y}$? We clarify below.

We noted that a factor graph is defined as a product of functions, rather than normalised distributions. The joint factor graph over both observed $\tilde{\mathbf{x}}$ and unobserved $\mathbf{x}$ variables

$$p(\mathbf{x}, \tilde{\mathbf{x}} = \mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c, \tilde{\mathbf{x}}_c = \mathbf{y}_c) \tag{2.35}$$

is proportional to the posterior distribution

$$p\left(\mathbf{x}|\tilde{\mathbf{x}}=\mathbf{y}\right)=\frac{1}{Z_{\mathbf{x}|\mathbf{y}}\left(\mathbf{y}\right)}\prod_{c\in\mathcal{C}}\phi_c\left(\mathbf{x}_c,\tilde{\mathbf{x}}_c=\mathbf{y}_c\right)=\frac{Z}{Z_{\mathbf{x}|\mathbf{y}}\left(\mathbf{y}\right)}p\left(\mathbf{x},\tilde{\mathbf{x}}=\mathbf{y}\right). \tag{2.36}$$

Thus, if we do not account for normalisation, the joint and the posterior are equivalent, and BP provides an estimate of posterior marginals $p\left(x_i|\mathbf{y}\right)$.

This raises the question of *why* we don't need to account for normalisation when doing marginal inference. The answer is that, in general, we assume that either the factors in our model or their outgoing messages are of a known parametric family of distributions. Therefore the marginal beliefs (2.30) are scalar distributions also of a known parametric form. Thus, if we estimate a marginal distribution up to a constant, normalisation can be recovered post-hoc via a low-dimensional, easily computable summation/integral.

### 2.4.3 Loopy Belief Propagation

In Section 2.4.1, we derived BP for tree-structured models, which relied upon the observation that a tree can be divided into unconnected subtrees branching out from a target node $x_i$. For cyclic graphs, however, these subgraphs are connected and exchange messages, so the messages from the subgraphs to the target variable $\{m_{\phi_k\to x_i}\left(x_i\right)\}_{\phi_k\in\mathrm{nei}(x_i)}$ are no longer isolated from one another.

There are many approaches to deal with inference in loopy graphs. For example, the junction-tree algorithm [JTA; Lauritzen and Spiegelhalter, 1988] combines groups of scalar variables into vector-valued variables until the resulting graph with higher order nodes forms a tree in which BP is exact. Further, Tree-reweighted BP [TRBP; Wainwright et al., 2003a] decomposes the loopy graph as a convex combination of spanning trees and executes a message passing algorithm in the spanning trees which optimises a convex upper bound on the partition function. However, both come with significant computational overhead during inference: the JTA due to more highly connected factors to marginalise and TRBP due to keeping track of the messages from multiple spanning trees. Techniques such as belief optimisation [Welling and Teh, 2001], directly minimise the Bethe free energy (whose stationary points are the fixed points of loopy BP), however they are not widely used due to their implementation difficulty and slower rate of convergence [Elidan et al., 2006, Sutton and McCallum, 2012].

One alternative is to "do it anyway", and apply the rules of BP to loopy models, knowing

that the resulting inference will not be exact. This is known as loopy BP and is a popular method of inference despite its approximate nature. It became widely known after highly successful decoding algorithms for low-density parity check codes [Gallager, 1962] and turbo codes [Berrou et al., 1993] were shown to be instances of BP in loopy graphical models [MacKay, 2002, McEliece et al., 1998]. It has since been shown to work well empirically across many other domains [see e.g. Frey et al., 2001, Murphy et al., 1999, Wang et al., 2015, Ortiz et al., 2020, 2022, Murai et al., 2023, Patwardhan et al., 2022].

#### 2.4.3.1 Heuristics

There are a number of noteworthy heuristics in relation to the convergence of loopy BP [see e.g. Box 11.B of Koller and Friedman, 2009]. First, it is common to apply damping to the factor-to-variable message updates [Murphy et al., 1999]

$$m_{\phi_k \to x_i}(x_i) \leftarrow \lambda m_{\phi_k \to x_i}(x_i) + (1-\lambda) \sum_{\mathbf{x}_{k \setminus i}} \phi_k(\mathbf{x}_k) \prod_{x_j \in \text{nei}(\phi_k) \setminus x_i} m_{x_j \to \phi_k}(x_j), \tag{2.37}$$

for $\lambda \in [0, 1)$. This interpolates the message update between the standard message update (2.29) and the value of the message from the previous iteration. This has been shown to reduce the prevalence of oscillations, while ensuring that if convergence does occur the fixed point will also be a fixed point of the undamped system [Murphy et al., 1999].

The scheduling of messages is also known to affect convergence. In comparison to synchronous message updates, in which all messages are updated in parallel, asynchronous schedules often enjoy better convergence [Koller and Friedman, 2009, Elidan et al., 2006, Wainwright et al., 2003b]. Figure 11.C.1 of Koller and Friedman [2009] shows that a simple asynchronous schedule converges faster and more often than synchronous message passing for a small Ising model. More sophistacated schedules include tree parameterisation of Wainwright et al. [2003b], who select a set of spanning trees and do message update sweeps through each of them one at a time, enabling fast information propagation across the whole model. Dynamic scheduling, such as residual BP [Elidan et al., 2006], has been shown to outperform tree-based schedules however. Residual BP works by prioritising further updates for messages whose two most recent updates differed by the largest margin, i.e. those for which recent updates causes the biggest change.

Last, Koller and Friedman [2009] notes that non-convergence in loopy BP is often a local problem, in the sense that most of the marginal estimates converge, but a small

subset oscillate. In this case, good performance may be achieved by simply terminating the inference after a particular number of iterations and use the resulting belief estimates.

#### 2.4.3.2 Variational Interpretation

Yedidia et al. [2005] provide a different perspective on loopy BP, showing it can be viewed as variational inference rather than simply heuristic. In variational inference, we approximate an intractable $p(\mathbf{x}) = \exp(-E(x))/Z$ with a tractable distribution $q(\mathbf{x})$ which we optimise to match $p(\mathbf{x})$ as closely as possible under the *variational free energy*,

$$F(q) = U(q) - H(q) \tag{2.38}$$

$$U(q) := \mathbb{E}_{q(x)}[E(\mathbf{x})], \quad \text{(average energy)} \tag{2.39}$$

$$H(q) := -\mathbb{E}_{q(x)}[\log q(\mathbf{x})] \quad \text{(entropy).} \tag{2.40}$$

By substituting $E(x) = -\log Z - \log p(\mathbf{x})$ we can show that

$$F(q) = -\log Z + D_{\mathrm{KL}}(q\|p) \tag{2.41}$$

where $D_{\mathrm{KL}}(r\|s)$ is the Kullback–Leibler divergence from $r$ to $s$. As $Z$ is constant wrt $q$, minimising $F$ is equivalent to minimising $D_{\mathrm{KL}}(q\|p)$, i.e. moving $q(\mathbf{x})$ to be as close as possible to $p(\mathbf{x})$. $D_{\mathrm{KL}}(q\|p)$ is non-negative and is zero iff $q = p$.

Much of variational inference research is focused on designing $q(\mathbf{x})$ to be sufficiently flexible to match the important characteristics of $p(\mathbf{x})$ while maintaining computational feasibility. One of the most straightforward approximations is the mean-field distribution

$$q_{\mathrm{MF}}(\mathbf{x}) := \prod_{i=1}^{N_x} q_i(x_i) \tag{2.42}$$

however, this is known to be restrictive due to its inability to capture dependencies between variables.

The principle behind the *Bethe approximation* presented by Yedidia et al. [2005] is to use a tree-structured family of approximating distributions, even if target $p(\mathbf{x})$ contains cycles. Any tree-structured joint distribution can be decomposed via sum (2.2) and product

(2.1) rules as

$$q_{\text{Bethe}}(\mathbf{x}) = \frac{\prod_{c\in\mathcal{C}} q_c(\mathbf{x}_c)}{\prod_{i=1}^{N_x} q_i(x_i)^{d_i-1}} \tag{2.43}$$

[Cowell, 1998] where $d_i$ is the *degree* of $x_i$, i.e. the number of factors it is connected to. Substituting $q = q_{\text{Bethe}}$ into the average energy (2.39) and entropy (2.40) definitions for a factor graph $p(x) = \left(\prod_c \phi_c(\mathbf{x}_c)\right)/Z$ gives

$$U(q_{\text{Bethe}}) = -\sum_{c\in\mathcal{C}} \mathbb{E}_{q_c(\mathbf{x}_c)}\left[\log \phi_c(\mathbf{x}_c)\right], \tag{2.44}$$

$$H(q_{\text{Bethe}}) = -\sum_{c} \mathbb{E}_{q_c(\mathbf{x}_c)}\left[\log q_c(\mathbf{x}_c)\right] + \sum_{i=1}^{N_x} \mathbb{E}_{q_i(x_i)}\left[\log q_i(x_i)\right], \tag{2.45}$$

which defines the *Bethe free energy*, $F_{\text{Bethe}} := U(q_{\text{Bethe}}) - H(q_{\text{Bethe}})$. In order to have valid marginals and clique distributions, we consider the *constrained* Bethe free energy, i.e. $F_{\text{Bethe}}$ for $q$, where $q$ satisfies

$$\sum_{x_i} q_i(x_i) = 1 \qquad \text{(marginal normalisation)} \tag{2.46}$$

$$\sum_{\mathbf{x}_c} q_c(\mathbf{x}_c) = 1 \qquad \text{(clique normalisation)} \tag{2.47}$$

$$\sum_{\mathbf{x}_{c\setminus j}} q_c(x_j, \mathbf{x}_{c\setminus j}) = q_j(x_j) \qquad \text{(local marginal consistency).} \tag{2.48}$$

Enforcing these constraints with Lagrange multipliers $\{\alpha_i\}, \{\beta_c\}, \{\gamma_{cj}\}$, we arrive at the following Lagrangian for the constrained Bethe free energy

$$\mathcal{L}_{\text{Bethe}}(q) = F_{\text{Bethe}}(q) \tag{2.49}$$

$$+ \sum_{i=1}^{N_x} \alpha_i \left(\sum_{x_i} q_i(x_i) - 1\right) \tag{2.50}$$

$$+ \sum_{c\in\mathcal{C}} \beta_c \left(\sum_{\mathbf{x}_c} q_c(\mathbf{x}_c) - 1\right) \tag{2.51}$$

$$+ \sum_{c\in\mathcal{C}} \sum_{x_j\in\text{nei}(\phi_c)} \gamma_{cj} \left(\sum_{\mathbf{x}_{c\setminus j}} q_c(x_j, \mathbf{x}_{c\setminus j}) - q_j(x_j)\right). \tag{2.52}$$

At the zero-gradient points of $\mathcal{L}_{\text{Bethe}}$ wrt $q_i(x_i)$ and $q_c(\mathbf{x}_c)$ we have

$$q_i^*(x_i) = \exp\left(\frac{1}{d_i - 1}\left(1 - \alpha_i + \sum_{\phi_c \in \text{nei}(x_i)} \gamma_{ci}\right)\right) \tag{2.53}$$

$$q_c^*(\mathbf{x}_c) = \phi_c(\mathbf{x}_c) \exp\left(\beta_c - 1 + \sum_{x_k \in \text{nei}(\phi_c)} \gamma_{ck}\right). \tag{2.54}$$

If we then assign multiplier $\gamma_{cj}$ to be the following function of the factor-to-variable messages

$$\gamma_{cj} = \log \prod_{\phi_e \in \text{nei}(x_j) \setminus \phi_c} m_{\phi_e \to x_j}(x_j), \tag{2.55}$$

we find (2.53) and (2.54) follow

$$q_i^*(x_i) \propto \prod_{\phi_c \in \text{nei}(x_i)} m_{\phi_c \to x_i}(x_i) \tag{2.56}$$

$$q_c^*(\mathbf{x}_c) \propto \phi_c(\mathbf{x}_c) \prod_{x_j \in \text{nei}(\phi_c)} \prod_{\phi_e \in \text{nei}(x_j) \setminus \phi_c} m_{\phi_e \to x_j}(x_j). \tag{2.57}$$

Again applying the normalisation and marginalisation constraints (2.46), (2.47), (2.48), we see that (2.56) is the marginal belief estimate (2.30) and the marginalisation of (2.57) is the factor-to-variable message update (2.29) with the variable-to-factor messages (2.28) folded in. We thus see that the fixed points of BP correspond to stationary points of the Bethe free energy.

This variational interpretation is helpful for understanding the differing accuracy of BP for tree and cyclic models. If the model $p(\mathbf{x})$ is tree-structured, and therefore within the approximating family $q_{\text{Bethe}}(\mathbf{x})$, our search can recover the exact marginals $q_i(x_i) = p_i(x_i)$. However, the loopy models are outside $q_{\text{Bethe}}(\mathbf{x})$, and therefore loopy BP can only find approximate marginals.

### 2.4.4 Gaussian Belief Propagation

We now review BP in Gaussian models: GBP. We first derive the specific update rules for Gaussian factors and then discuss theoretical guarantees.

As discussed in Section 2.4.1.1, BP message updates are more efficient using the natural

parameterisation of the factors. To find this for a Gaussian distribution, we start with the "moment" form

$$\phi_c(\mathbf{x}_c) \propto \mathcal{N}(\mathbf{x}_c; \mu_c, \Sigma_c) = \frac{1}{|2\pi\Sigma_c|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}_c - \mu_c)^\intercal \Sigma_c^{-1}(\mathbf{x}_c - \mu_c)\right) \tag{2.58}$$

$$= \frac{1}{Z(\eta_c, \Lambda_c)} \exp\left(-\frac{1}{2}\mathbf{x}_c^\intercal \Lambda_c \mathbf{x}_c + \eta_c^\intercal \mathbf{x}_c\right) = \mathcal{N}^{-1}(\mathbf{x}_c; \eta_c, \Lambda_c) \tag{2.59}$$

$$\eta_c := \Sigma_c^{-1}\mu_c \tag{2.60}$$

$$\Lambda_c := \Sigma_c^{-1} \tag{2.61}$$

where $\mathcal{N}^{-1}$ denotes the natural, "canonical" or "information" form of the multivariate Gaussian, $\eta_c$ is known as the *information* vector, $\Lambda_c$ is the *precision* matrix. We can see this follows the natural parameterisation of exponential family distributions (2.33), with $\nu = [\eta_c^\intercal, \mathrm{vec}\left(\Sigma_c^{-1}\right)^\intercal]^\intercal$ and $T(\mathbf{x}_c) = [\mathbf{x}_c^\intercal, \mathrm{vec}\left(\mathbf{x}_c\mathbf{x}_c^\intercal\right)^\intercal]^\intercal$.

Gaussian distributions are closed under both marginalisation and conditioning. Thus, BP messages from Gaussian factors will also be Gaussian distributed. The general form of the variable-to-factor message updates is

$$m_{x_i \to \phi_k}(x_i) = \prod_{\phi_l \in \mathrm{nei}(x_i) \setminus \phi_k} m_{\phi_l \to x_i}(x_i) \tag{2.62}$$

which for Gaussian messages with natural parameterisation gives the straightforward parameter sums

$$\eta_{x_i \to \phi_k} = \sum_{\phi_l \in \mathrm{nei}(x_i) \setminus \phi_k} \eta_{\phi_k \to x_i} \tag{2.63}$$

$$\Lambda_{x_i \to \phi_k} = \sum_{\phi_l \in \mathrm{nei}(x_i) \setminus \phi_k} \Lambda_{\phi_k \to x_i}. \tag{2.64}$$

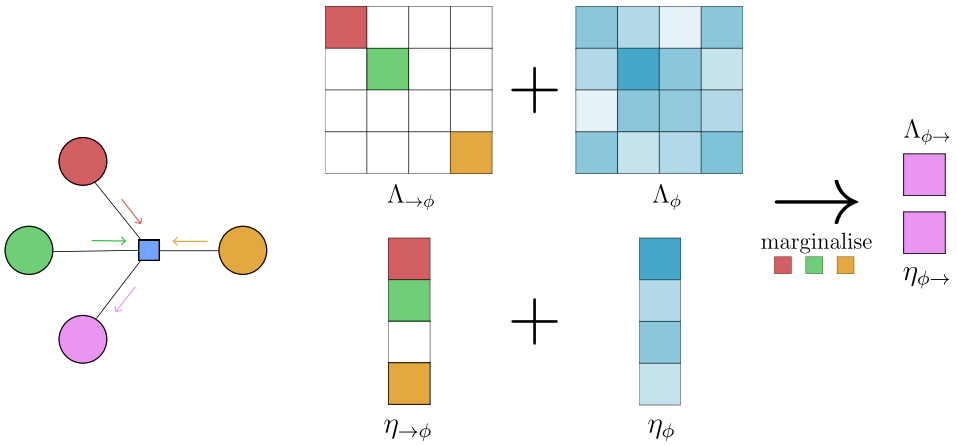



Figure 2.4: **The mechanics of GBP factor-to-variable updates.** All incoming messages to the factor, excluding the message from the recipient, are added to the factor parameters. The resulting distribution $\mathcal{N}^{-1}\left(\eta_{\to\phi}+\eta_{\phi},\, \Lambda_{\to\phi}+\Lambda_{\phi}\right)$ is marginalised of all variables except the recipient to give the outgoing message.

The factor-to-variable message update is more involved. As in the general form (2.29) we take the product of the factor value with incoming messages except that from the recipient variable. We then marginalise all variables except the recipient. For Gaussian messages this gives

$$m_{\phi_k\to x_i}(x_i) = \int \mathcal{N}^{-1}\left(\mathbf{x}_k; \eta_k + \sum_{x_j\in \mathrm{nei}(\phi_k)\setminus x_i} \eta_{x_j\to\phi_k}\,,\; \Lambda_k + \sum_{x_j\in \mathrm{nei}(\phi_k)\setminus x_i} \Lambda_{x_j\to\phi_k}\right) \mathrm{d}\mathbf{x}_{k\setminus i}. \tag{2.65}$$

While conditioning is efficient in the natural parameterisation, marginalisation is comparatively expensive. Applying the rules of marginalisation of Gaussians in the natural parameterisation, we find

$$\eta_{\phi_k\to x_i} = \eta_k^{(i)} - \Lambda_{k+m_{\setminus i}}^{(i,\setminus i)} \left(\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)}\right)^{-1} \eta_{k+m_{\setminus i}}^{(\setminus i)} \tag{2.66}$$

$$\Lambda_{\phi_k\to x_i} = \Lambda_k^{(i,i)} - \Lambda_{k+m_{\setminus i}}^{(i,\setminus i)} \left(\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)}\right)^{-1} \Lambda_{k+m_{\setminus i}}^{(\setminus i,i)}, \tag{2.67}$$

where superscript $^{(i)}$ denotes the indices of the factor parameters corresponding to $x_i$, and vice versa for $^{(\setminus i)}$. Subscript $_{k+m_{\setminus i}}$ denotes the parameter resulting from adding the factor parameter $\eta_k, \Lambda_k$ to all incoming message parameters except the message from $x_i$.

We illustrate the factor-to-variable update in Figure 2.4. The incoming message precisions

are compiled into a diagonal matrix with a zero in the element corresponding to the recipient variable. Similarly, the incoming information elements are stacked into a vector. These combined messages are added to the factor parameters, and the resulting multivariate Gaussian is marginalised of all but the recipient variable to give the outgoing message.

As the variable-to-factor updates is a simple summation over incoming messages, is outgoing message has time complexity $\mathcal{O}\left(|\text{nei}(x_i)|\right)$. However, as described in Section 2.4.1.1, by first computing the belief $p\left(x_i\right)$ and then subtracting factor-to-variable messages, we can reduce the complexity of *all* message updates from a variable to $\mathcal{O}\left(|\text{nei}(x_i)|\right)$. The worst-case complexity for factor-to-variable messages however is $\mathcal{O}\left(|\text{nei}(\phi_k)|^3\right)$ due to the inversion of $\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)} \in \mathbb{R}^{|\text{nei}(\phi_k)-1|\times|\text{nei}(\phi_k)-1|}$. We will see in Chapter 4 that $\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)}$ may be low-rank which can be exploited via the Woodbury identity for faster inversion.

#### 2.4.4.1 Convergence

For tree-structured graphs, BP in Gaussian models is exact as in the general case. Similarly, GBP is not guaranteed to converge in loopy models. However, Weiss and Freeman [1999] show that if GBP does converge in models with cycles, it is guaranteed that the estimated marginal means are exact, though the marginal variances are overconfident. They also provide a condition for convergence: the global precision matrix of the model must be diagonally dominant, i.e.

$$|\Lambda_{i,i}| > \sum_{j\neq i} |\Lambda_{i,j}|, \quad i = 1, 2, \ldots, N_x. \tag{2.68}$$

Malioutov et al. [2006] improve upon diagonal dominance, using the notion of *walk-summability*,

$$\rho\left(I - \Lambda^{(n)}\right) < 1 \tag{2.69}$$

where $\rho\left(\cdot\right)$ returns the maximum eigenvalue of the given matrix, and $\Lambda^{(n)}$ denotes the normalised precision matrix, for which variables have been rescaled to ensure $\Lambda_{ii}^{(n)} = 1$. They show that walk-summability is a more general condition for convergence than diagonal dominance. Johnson et al. [2009] present an iterative preconditioning scheme in which walk-summability is guaranteed at each stage, thus guaranteeing convergence of loopy GBP overall. Unfortunately, it introduces significant computational overhead as it requires solving

a sequence of models rather than a single model.

#### 2.4.4.2 Non-linear Factors

In some cases, Gaussian factors may be a poor description of the data we are modelling, but we may want to retain the computational benefits of GBP. We can do so by making Gaussian approximations to models which are not exactly Gaussian. Two notable approaches to achieve this are i) linearisation based on local curvature as described in Davison and Ortiz [2019], and ii) moment-matching of the outgoing messages [Minka, 2013]. In this thesis we use the local linearisation for convenience of implementation, but we discuss the merits of both in Section 2.4.5.

We now derive the form of the parameters $\eta, \Lambda$ for factors approximated as Gaussian, following the derivation of Davison and Ortiz [2019]. We assume a factor with observations $\mathbf{y}_k$ follows the general form

$$\phi_k\left(\mathbf{x}_k|\mathbf{y}_k\right) = \exp\left(-E_k\left(\mathbf{x}_k|\mathbf{y}_k\right)\right), \tag{2.70}$$

$$E_k\left(\mathbf{x}_k|\mathbf{y}_k\right) = \frac{1}{2}\left(\mathbf{y}_k - h_k\left(\mathbf{x}_k\right)\right)^\intercal \Lambda_{\mathbf{y}_k}\left(\mathbf{y}_k - h_k\left(\mathbf{x}_k\right)\right) \tag{2.71}$$

where $h_k\left(\cdot\right)$ is known as the *measurement function*, a prediction of the observations $\mathbf{y}_k$ based on the current estimate of the state $\mathbf{x}_k$ and $\Lambda_{\mathbf{y}_k}$ is the observation precision. While this assumption might seem restrictive, we note that it accommodates any normalised function, $\phi_k\left(\mathbf{x}_k\right) = g_k\left(\mathbf{x}_k\right),\ 0 < g\left(\mathbf{x}_k\right) < 1$, by setting $\mathbf{y}_k = 0$, $\Lambda_{\mathbf{y}_k} = 1$, $h_k\left(\mathbf{x}_k\right) = \sqrt{-2\log g_k\left(\mathbf{x}_k\right)}$.

Factor (2.70) is Gaussian (over $\mathbf{x}_k$) if $h_k\left(\cdot\right)$ is linear. We can therefore approximate the factor as Gaussian by approximating $h_k\left(\cdot\right)$ as linear. To do so, we expand $h_k\left(\mathbf{x}_k\right)$ about a chosen *linearisation point* $\mathbf{x}_{k,0}$:

$$h_k\left(\mathbf{x}_k\right) \approx h_k\left(\mathbf{x}_{k,0}\right) + J_{k,0}\left(\mathbf{x}_k - \mathbf{x}_{k,0}\right) \tag{2.72}$$

$$J_{k,0} := \left.\frac{\partial h\left(\mathbf{x}_k\right)}{\partial \mathbf{x}_k}\right|_{\mathbf{x}_{k,0}}. \tag{2.73}$$

If we substitute this back into (2.70) we find a factor of Gaussian form with energy

$$E_{k,\text{Gauss}}\left(\mathbf{x}_k|\mathbf{y}_k\right) = \frac{1}{2}\left(\mathbf{y}_k - h_k\left(\mathbf{x}_{k,0}\right) - J_{k,0}\left(\mathbf{x}_k - \mathbf{x}_{k,0}\right)\right)^\intercal \Lambda_{\mathbf{y}_k}\left(\mathbf{y}_k - h_k\left(\mathbf{x}_{k,0}\right) - J_{k,0}\left(\mathbf{x}_k - \mathbf{x}_{k,0}\right)\right) \tag{2.74}$$

$$\approx E_k\left(\mathbf{x}_k\right) \tag{2.75}$$

whose parameters we see by comparison to the stated natural parameterisation for Gaussians (2.59)

$$\eta_k = J_{k,0}^\intercal \Lambda_{\mathbf{y}_k}\left(\mathbf{y}_k - h_k\left(\mathbf{x}_{k,0}\right) + J_{k,0}\mathbf{x}_{k,0}\right) \tag{2.76}$$

$$\Lambda_k = J_{k,0}^\intercal \Lambda_{\mathbf{y}_k} J_{k,0}. \tag{2.77}$$

We note that the Jacobian $J_{k,0}$ scales the message precision, such that messages from regions linearised about a high-gradient region of $h\left(\cdot\right)$ will be stronger than flatter regions. This is intuitive — steeper regions of $h\left(\cdot\right)$ will result in less ambiguous predictions over a range of uncertain $\mathbf{x}_k$, such that the variables ought to then be better constrained by observation $\mathbf{y}_k$.

Linearisation admits the computation of approximate $\eta_k$ and $\Lambda_k$ which can be used for factor-to-variable message updates (2.66) and (2.67), but we have not yet defined how the linearisation point $\mathbf{x}_{k,0}$ should be chosen. Typically we linearise the factor around the current MAP estimate, i.e. $x_i^* = \text{argmax}\, p\left(x_i|\mathbf{y}\right)$, alternating between i) running a few iterations of GBP to update $\mathbf{x}_k^*$ given fixed $\mathbf{x}_{k,0}$, and ii) updating $\mathbf{x}_{k,0} \leftarrow \mathbf{x}_k^*$, and relinearising $\phi_k$. This is somewhat heuristic and choosing the number of GBP iterations between updates, and the initialisation of the linearisation points, is typically determined empirically.

**Robust Factors** Assumed Gaussianity of residuals is reasonable in many settings, but may result in poor inference in cases where the true data are heavy tailed. Instead, M-estimators may be more robust. Davison and Ortiz [2019] introduce a technique for GBP in models with such robust factors. In contrast to many robust estimator implementations which apply the above linearisation scheme to the robustified density [e.g. Agarwal et al., 2012], Davison and Ortiz [2019] take inspiration from Agarwal et al. [2013] and dynamically scale the precision of the factor based on the energy of its current configuration. A consequence of this method is that robust factors with flat tails will always send messages with non-zero precision, which is not true for the linearisation approach.

Consider a robust factor with potential

$$\phi_{k,\text{rob}}(\mathbf{x}_k) = \exp\left(-E_{k,\text{rob}}(\mathbf{x}_k)\right). \tag{2.78}$$

The approach of Davison and Ortiz [2019] asks *what are the parameters of a Guassian factor with equal energy to* $\phi_{k,\text{rob}}$ *at the current MAP* $\mathbf{x}_k^*$? Specifically, they rescale covariance by a factor $1/c(\mathbf{x}_k^*)$, such that the resulting Gaussian has energy equal to $E_{k,\text{rob}}$ at $\mathbf{x}_k^*$. Rescaling the covariance by $1/c(\mathbf{x}_k^*)$ rescales the energy by $c(\mathbf{x}_k^*)$, so we can write

$$E_{k,\text{rob}}(\mathbf{x}_k^*) = c(\mathbf{x}_k^*)\, E_{k,\text{Gauss}}(\mathbf{x}_k^*) \tag{2.79}$$

$$\implies c(\mathbf{x}_k^*) = \frac{E_{k,\text{rob}}(\mathbf{x}_k^*)}{E_{k,\text{Gauss}}(\mathbf{x}_k^*)}. \tag{2.80}$$

Noting that scaling the covariance $\Sigma_k \to \Sigma_k / c(\mathbf{x}_k^*)$ is equivalent to $\eta_k \to c(\mathbf{x}_k^*)\,\eta_k$, $\Lambda_k \to c(\mathbf{x}_k^*)\,\Lambda_k$, we can implement the robustification by multiplying the factor parameters by the ratio of robust and Gaussian energies.

For example, a Huber robust factor has energy

$$E_{k,\text{Huber}}(\mathbf{x}_k) = \begin{cases} E_{k,\text{Gauss}}(\mathbf{x}_k) & E_{k,\text{Gauss}}(\mathbf{x}_k) < E_{k,\text{thresh}} \\ 2\sqrt{E_{k,\text{thresh}} E_{k,\text{Gauss}}(\mathbf{x}_k)} - E_{k,\text{thresh}} & \text{otherwise} \end{cases} \tag{2.81}$$

where $E_{k,\text{thresh}}$ is a hyperparameter which determines the location of the transition from Gaussian to heavy tailed. The adjustment factor at $\mathbf{x}_k^*$ is therefore

$$c(\mathbf{x}_k^*) = \begin{cases} 1 & E_{k,\text{Gauss}}(\mathbf{x}_k^*) < E_{k,\text{thresh}} \\ 2\frac{\sqrt{E_{k,\text{thresh}}}}{\sqrt{E_{k,\text{Gauss}}(\mathbf{x}_k^*)}} - \frac{E_{k,\text{thresh}}}{E_{k,\text{Gauss}}(\mathbf{x}_k^*)} & \text{otherwise.} \end{cases} \tag{2.82}$$

In the heavy tailed region $E_{k,\text{Gauss}}(\mathbf{x}_k^*) > E_{k,\text{thresh}}$, meaning $c(\mathbf{x}_k^*) < 1$. This will reduce the factor precision and result in weaker messages being sent from the factor. This is desirable behaviour for a robust factor in the presence of outliers.

### 2.4.5 Relation to Expectation Propagation

Expectation propagation [EP; Minka, 2013], like BP, is a message passing algorithm. It is a method of approximate inference which breaks down the complexity of inference by

approximating a single factor at each step. In this section we will show that EP is a particular form of BP in which the factor-to-variable messages are approximated as an assumed functional form via moment matching. In contrast, the linearisation approach described in Section 2.4.4.2 approximates the factors directly.

In EP algorithm we approximate the true distribution, which may be written as a factor graph,

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c) \tag{2.83}$$

with another distribution of the same structure but with tractable factors $\tilde{\phi}_c$

$$q(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \tilde{\phi}_c(\mathbf{x}_c). \tag{2.84}$$

The algorithm repeats the following steps until convergence [Murphy, 2012, Section 22.5.2]:

1. Choose a factor $\tilde{\phi}_k(\mathbf{x}_k)$ to update
2. Remove the factor from the current approximation

$$q^{\setminus k}(\mathbf{x}) = \frac{q(\mathbf{x})}{\tilde{\phi}_k(\mathbf{x}_k)} \tag{2.85}$$

3. Compute the new approximation by solving

$$\text{argmin}_{q_{\text{new}}(\mathbf{x})} D_{\text{KL}}\left(\frac{1}{Z_k}\phi_k(\mathbf{x}_k)\, q^{\setminus k}(\mathbf{x}) \middle\| q_{\text{new}}(\mathbf{x})\right) \tag{2.86}$$

   which can be done by setting the moments of $q_{\text{new}}(\mathbf{x})$ to those of $\phi_k(\mathbf{x}_k)\, q^{\setminus k}(\mathbf{x})$ for exponential family distributions.

4. Compute the corresponding approximating factor as it will be removed during subsequent updates

$$\tilde{\phi}_k(\mathbf{x}_k) = Z_k \frac{q_{\text{new}}(\mathbf{x})}{q^{\setminus k}(\mathbf{x})} \tag{2.87}$$

At each step, the approximation of a single factor $\phi_k(\mathbf{x})$ is updated. The update target

combines the ground truth $\phi_k(\mathbf{x})$ with the current state of the other factors $\prod_{l\setminus k}\tilde{\phi}_l(\mathbf{x}_l)$ (rather than updating $\tilde{\phi}_k(\mathbf{x}_k)$ based on $\phi_k(\mathbf{x})$ in isolation) for better approximation accuracy.

It is clear that EP shares a similar local character to BP in the sense that each update pertains to a local region of the graph, but let us explore their relationship more precisely, following the exposition of [Bishop, 2006, Section 8.4.4]. Consider a fully factorised approximating family

$$\tilde{\phi}_c(\mathbf{x}_c) = \prod_{x_i\in\mathrm{nei}(\phi_c)} \tilde{\phi}_{ci}(x_i) \tag{2.88}$$

$$q^{\setminus c}(\mathbf{x}) = \prod_{d\in\mathcal{C}\setminus c}\prod_{x_j\in\mathrm{nei}(\phi_d)} \tilde{\phi}_{dj}(x_j)\,. \tag{2.89}$$

Suppose we want to refine $\tilde{\phi}_{ci}(x_i)$. To do so we must compute the $x_i$ marginal of

$$\phi_c(\mathbf{x}_c)\prod_{d\in\mathcal{C}\setminus c}\tilde{\phi}_d(\mathbf{x}_d) = \phi_c(\mathbf{x}_c)\prod_{d\in\mathcal{C}\setminus c}\prod_{x_j\in\mathrm{nei}(\phi_d)}\tilde{\phi}_{dj}(x_j) \tag{2.90}$$

and then divide by $q^{\setminus c}(\mathbf{x})$. This division causes terms which depend on other factors to cancel out, and we end up with the updated approximating factor

$$\tilde{\phi}_{ci}(x_i) \propto \int \phi_c(\mathbf{x}_c)\prod_{x_j\in\mathbf{x}_c\setminus x_i}\prod_{\phi_d\in\mathrm{nei}(x_j)\setminus\phi_c}\tilde{\phi}_{dj}(x_j)\,\mathrm{d}\mathbf{x}_{c\setminus i} \tag{2.91}$$

which is equal to the factor-to-variable message update in BP (2.26), where EP factors correspond to BP messages $\tilde{\phi}_{ci}(x_i) = m_{\phi_c\to x_i}(x_i)$.

As we have discussed in Section 2.4.4.2, many problems require models with nonlinear $\phi_c(\mathbf{x}_c)$. In this case EP approximates the product $\phi_c(\mathbf{x}_c)\,q^{\setminus c}(\mathbf{x})$ as linear-Gaussian via moment matching (point 3 above), i.e. with a Gaussian whose parameters are

$$\mu_c \propto \int \mathbf{x}_c\phi_c(\mathbf{x}_c)\,q^{\setminus c}(\mathbf{x})\,\mathrm{d}\mathbf{x}_c \tag{2.92}$$

$$\Sigma_c \propto \int \mathbf{x}_c\mathbf{x}_c^\intercal\phi_c(\mathbf{x}_c)\,q^{\setminus c}(\mathbf{x})\,\mathrm{d}\mathbf{x}_c. \tag{2.93}$$

This may be arguably be a more accurate approximation than factor linearisation (Section 2.4.4.2) as it estimates the parameters via a global average, rather than by the local curvature. However, it has two drawbacks. First, for nonlinear $\phi_c(\mathbf{x}_c)$, the moment calcula-

tions (2.92) and (2.93) must be derived analytically. They may be intractable and require approximation. In contrast, linearisation requires only that the gradient of the measurement function $h(\cdot)$ is defined, and the resulting Jacobians can be computed automatically via e.g. automatic differentiation frameworks. Second, moment matching optimises the "forward" KL divergence $D_{\mathrm{KL}}(p\|q)$ which is known to result in mode-averaging behaviour. When approximating complex multi-modal distributions, this can lead to $q$ having significant density between modes of $p$, where $p(\mathbf{x}) \approx 0$.

## 2.5 Gaussian Process Regression

A Gaussian process (GP) [Williams and Rasmussen, 2006] is a generalisation of a Gaussian distribution to the infinite dimensional space of functions. It is defined by a mean function $\mu : \mathbb{R}^D \to \mathbb{R}$, and a covariance (or kernel) function $k_\theta : \mathbb{R}^{D\times D} \to \mathbb{R}$, which determines the correlation between two function values given their input locations. $k_\theta(\cdot,\cdot)$ has hyperparameters $\theta$. Mathematically, we denote a GP over $f : \mathbb{R}^D \to \mathbb{R}$ as

$$f \sim \mathcal{GP}(\mu, k_\theta). \tag{2.94}$$

For regression, a GP is used as a prior $p(f(\cdot)|\theta)$. For an iid Gaussian likelihood from observations $\mathbf{y} = [y_1, \ldots, y_N]^\intercal$ at inputs $X = [\mathbf{x}_1^\intercal, \ldots, \mathbf{x}_N^\intercal]^\intercal$, the posterior prediction $f(\cdot)$ at test point $\mathbf{x}^*$ is

$$p(f(\mathbf{x}^*)|\mathbf{y}) = \mathcal{N}(\mu_*, \Sigma_*) \tag{2.95}$$

$$\mu_* := \mathbf{k}_{x^*x}\left(K_x + \sigma^2 I_x\right)^{-1}\mathbf{y}, \tag{2.96}$$

$$\Sigma_* := K_{x^*} - \mathbf{k}_{x^*x}\left(K_x + \sigma^2 I_x\right)^{-1}\mathbf{k}_{xx^*}, \tag{2.97}$$

where $K_x \in \mathbb{R}^{N\times N}$ has elements $[K_x]_{ij} = k_\theta(\mathbf{x}_i, \mathbf{x}_j)$, $K_{x^*} \in \mathbb{R}^{1\times 1}$ has element $k_\theta(\mathbf{x}^*, \mathbf{x}^*)$, $\mathbf{k}_{xx^*}$ is a column vector of length $N$ with elements $[\mathbf{k}_{xx^*}]_i = k_\theta(\mathbf{x}_i, \mathbf{x}^*)$, and $\sigma$ is the observation noise. Kernel hyperparameters $\theta$ can be trained by gradient ascent of the (log-)marginal likelihood, $p(\mathbf{y}|\theta) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\theta)\mathrm{d}\mathbf{f}$, which trades-off model-fit with model complexity.

## 2.6 Conclusion

We thus conclude the overview of preliminary material. We have introduced probability theory and discussed how probabilities relate to beliefs. We have seen how a probabilistic model can be used to encode assumptions about how our observations were generated, and how the variables of interest relate to one another. They enable us to do inference via Bayes rule. PGMs are a useful representation of probabilistic models, which make concrete the independence relationships between variables. Of the PGM family, factor graphs of particular interest, as they avoid the need to represent problematic normalising constants.

Exact probabilistic inference is usually intractable, motivating approximate inference techniques. We discussed how MCMC can be used for sampling-based approximate inference, looking at Metropolis-Hastings and SGLD, before moving on to BP — a message passing algorithm for marginal inference. After deriving the algorithm for tree-structured models, we described how marginal inference in a factor graph relates to posterior inference. We then discussed loopy BP, the application of BP in graphs with cycles, for approximate marginal inference. The fixed points of BP are stationary points of the Bethe free energy functional, providing an interpretation of loopy BP from the perspective of variational inference. We then moved on to looking at BP in Gaussian models (GBP), first deriving the specific message updates, before considering its application to nonlinear models. We finished with a brief overview of GPs.

# Chapter 3

# Data Augmentation in Bayesian Neural Networks and the Cold Posterior Effect

We now discuss the work from our paper *Data Augmentation in Bayesian Neural Networks and the Cold Posterior Effect*, which was presented at *Uncertainty in Artificial Intelligence 2022* [Nabarro et al., 2022]. This paper was co-lead by Stoil Ganev who undertook the analysis of how our proposed data augmentation (DA) likelihood bounds affect non-Bayesian neural network performance (included in Appendix A.1). Further, Adrià Garriga-Alonso generated the kinetic temperature diagnostic plots included in Appendix A.2.

Many Bayesian deep learning works use DA during training. The standard approach to DA can be viewed as an amended log-likelihood function, which is averaged over the distribution of augmentations of a given input [Wenzel et al., 2020]. Izmailov et al. [2021] note that this "randomly perturbed log-likelihood does not have a clean interpretation as a valid likelihood function". In particular, the corresponding likelihood is unnormalised and the "constant" of normalisation depends on the network parameters. Further, the use of such DA coincides with the so called cold posterior effect (CPE), in which Bayesian neural network (BNN) posteriors only perform well with artificial "cooling", which concentrates density around their modes. This raises the question as to whether an invalid DA likelihood is responsible for the CPE.

Here, we provide several approaches to principled DA for BNNs via likelihood bounds

for DA-invariant models. We introduce a "finite orbit" setting which allows valid likelihoods to be computed exactly, and for the more usual "full orbit" setting we derive multi-sample bounds tighter than those used previously. However, we find that the CPE persists even in these principled models incorporating DA. This suggests that the CPE cannot be dismissed as an artefact of DA using incorrect likelihoods.

## 3.1 Introduction

The CPE [Wenzel et al., 2020] is the surprising observation that performance in neural networks (NNs) is not optimal when we use the usual Bayesian posterior given the dataset $\mathcal{D} = (X, \mathbf{y})$ [Kolmogorov, 1950, Savage, 1954, Jaynes, 2003]

$$p(\theta|\mathbf{y}, X) \propto p(\theta)\, p(\mathbf{y}|\theta, X) \tag{3.1}$$

where $\theta$ are the NN weights, $X$ is all inputs, and $\mathbf{y}$ is all outputs. Instead, we get better performance when using a "cold" posterior, i.e. the posterior taken to the power of $1/T$ where $T < 1$,

$$p_T(\theta|\mathbf{y}, X) \propto \left(p(\theta)\, p(\mathbf{y}|\theta, X)\right)^{\frac{1}{T}} . \tag{3.2}$$

Intuitively, lower $T$ makes the posterior more "peaked", concentrating the mass more tightly around the modes, and making larger modes more dominant (see e.g. Figure 1.3).

The origin of the CPE is highly contentious, with three leading potential explanations [Noci et al., 2021]. The first hypothesis is that the process of data curation for popular datasets such as CIFAR-10 and ImageNet [Krizhevsky et al., 2009, Deng et al., 2009] involves multiple annotators agreeing upon the label for each image. In that case, there are in effect multiple labels for each image, which inflates the likelihood (but not the prior) term causing a "cooler" posterior [Adlam et al., 2020, Aitchison, 2020]. Second, the prior is always misspecified, and prior misspecification is known to induce cold posterior-like effects in specific (non-neural network) models [Grünwald, 2012, Grünwald et al., 2017, Adlam et al., 2020], which might give an explanation for the CPE in NNs [Wenzel et al., 2020, Fortuin et al., 2022]. However, Fortuin et al. [2022] showed that better priors do not always reduce the size of the CPE, but can actually increase it. In particular, they found that incorporating spatial correlations in convolutional filters improved the performance of a

ResNet trained on CIFAR-10, but also increased the magnitude of the CPE. The third possible explanation is that the CPE is an artefact of DA [Wenzel et al., 2020, Izmailov et al., 2021], as DA gives a "randomly perturbed log-likelihood [which] does not have a clean interpretation as a valid likelihood function" [Izmailov et al., 2021]. This is supported by observations in which the CPE only exists with DA, and disappears without DA [Wenzel et al., 2020, Fortuin et al., 2022, Izmailov et al., 2021]. Of course it is quite possible that practical CPEs arise from a complex combination of these causes [Aitchison, 2020, Noci et al., 2021].

In spite of this controversy, much work on the CPE agrees that it is important to investigate integrating DA with Bayesian neural networks (BNNs), and to examine the interaction with the CPE. From Noci et al. [2021]: "It remains an interesting open problem how to properly account for data augmentation in a Bayesian sense." And from Izmailov et al. [2021]: "Data augmentation cannot be naively incorporated in the Bayesian neural network model." and "We leave incorporating data augmentation ... as an exciting direction of future work."

Perhaps the most common understanding of the interaction between the CPE and DA in BNNs is that DA increases the effective dataset size. From Izmailov et al. [2021]: "intuitively, data augmentation increases the amount of data observed by the model, and should lead to higher posterior contraction". From Osawa et al. [2019]: "DA increases the effective sample size". From, Noci et al. [2021]: "while data augmentation may increase the amount of data seen by the model, that increase is certainly not equal to the number of times each data point is augmented (after all, augmented data is not independent from the original data)."

In this work, we seek to understand whether the commonly used, but invalid DA likelihood can cause the CPE. Our contributions are as follows.

1. We give a formal argument that the notion that DA increases the effective dataset size is flawed (Section 3.2.1).

2. We motivate the need for multi-sample bounds, by showing that previous single-sample bounds on the likelihood are equivalent to averaging log-likelihoods, which is known to be problematic (3.21).

3. We derive a set of multi-sample lower bounds on the log-likelihood of a BNN incorporating DA (Section 3.2.2). These bounds are tighter than existing single-sample estimators for BNNs [e.g. Wenzel et al., 2020] and can be applied to a broad class of

likelihood functions [unlike Van der Wilk et al., 2018]. We discuss also the broader implications of the different bounds in Section 3.4.

4. We introduce a "finite orbit"[1] setting with a small number of admissible augmentations which allows us to compute *exact* log-likelihoods (Section 3.2.3).

5. We empirically evaluate the performance of the multi-sample bounds for BNN inference on image classification tasks (Section 3.3). In the latter case we explore the impact of both the bounds and the exact finite orbit likelihood on the CPE.

6. We find that the CPE persists even when using these principled DA likelihood bounds. This falsifies the hypothesis that the CPE is an artefact of loose bounds on the log-likelihood given by previous single-sample estimators.

We finish with some discussion summarising our findings and their reflection on the CPE (Section 3.6). Our conclusion in this work is that the CPE is not an artefact resulting from DA giving "randomly perturbed log-likelihood"s [Izmailov et al., 2021].

### 3.1.1 Notation

In the remainder of the chapter, we will follow Izmailov et al. [2021] in regarding models with loose, single-sample bounds as "unprincipled" (from Izmailov et al. [2021], the "randomly perturbed log-likelihood does not have a clean interpretation as a valid likelihood function"). In contrast, we term models using our exact log-likelihoods or our multi-sample bounds as being "principled".

For a set of input-output observations $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$, we will write the BNN log-posterior as

$$\log p\left(\theta|X, \mathbf{y}\right) = \log p\left(\theta\right) + \sum_{i=1}^{N} L^i(\theta) + \text{const}\,. \tag{3.3}$$

and throughout the chapter we will introduce different expressions for the log-likelihood terms $\{L^i\}_{i=1}^N$. First, we define the "no augmentation" log-likelihood as

$$L^i_{\text{noaug}}(y_i; \theta) = \log p\left(y_i|g(\mathbf{x}_i; \theta)\right). \tag{3.4}$$

[1] We employ the term "orbit" from group theory and function invariance [Kondor, 2008], even though our augmentations do not always form groups. In this work, it refers to the support of augmentation distribution $p(\mathbf{x}'|\mathbf{x})$.

where $g(\cdot;\theta)$ is a general NN. We denote DA samples as $\mathbf{x}'_i$, which are assumed to be generated by an input-conditional augmentation distribution, $p(\mathbf{x}'_i|\mathbf{x}_i)$. All expectations are over $p(\mathbf{x}'_i|\mathbf{x}_i)$. We focus on image classification, where $\mathbf{f}(\cdot;\theta)$ represents the output of a network parameterising the class logits, which relate to the vector of probabilities as

$$\mathbf{p}(\mathbf{x}_i;\theta) = \text{softmax}\,\mathbf{f}(\mathbf{x}_i;\theta)\,. \tag{3.5}$$

## 3.2 Method

### 3.2.1 Does DA Increase Dataset Size?

We first consider whether DA should be modelled as an increase in the number of training examples.

Many authors have claimed that DA increases the effective dataset size [Noci et al., 2021, Osawa et al., 2019, Izmailov et al., 2021]. Here we argue that this view leads to problems within the framework of probabilistic modelling. We can see this in the form of the resulting log-likelihood. For $K$ augmented inputs, $\mathbf{x}'_{i;k}$, we can write the log-likelihood for a single underlying input as,

$$L^i_{\text{add}}(y_i;\theta) = \sum_{k=1}^{K} \log p\left(y_i\middle|g(\mathbf{x}'_{i;k};\theta)\right). \tag{3.6}$$

For continuous transformations such as rotations, there are an infinite number of possible augmentations, $K=\infty$, so $L_{\text{add}}$ would result in the prior being ignored during inference. While this result seems strange, if the outputs for all augmentations $\mathbf{x}'_{i;k}$ were independently labelled (or if all the labels were correct) we would indeed have an infinitely large (conditionally) iid dataset and ignoring the prior would be the right answer. However, in practice, the unaugmented input $\mathbf{x}_i$ is labelled by an annotator who sometimes makes mistakes [Peterson et al., 2019], and the result $y_i$ is assumed to apply to all augmentations $\mathbf{x}'_{i;k}$. As such, the labels for different augmentations of the same input are not independent, and an approach (such as this one) which assumes they are cannot be valid.

A method which avoids having to specify the augmented dataset size is to average the log-likelihood over the augmentation distribution $p(\mathbf{x}'_i|\mathbf{x}_i)$

$$L^i_{\text{loss}}(y_i;\theta) = \mathbb{E}\left[\log p\left(y_i\middle|g(\mathbf{x}'_i;\theta)\right)\right]. \tag{3.7}$$

Indeed, most implementations which use DA when training BNNs target this log-likelihood, at least implicitly. They do so by taking a pre-existing inference algorithm and replacing the original input, $\mathbf{x}_i$, with a random augmentation, $\mathbf{x}'_i$. This approach is convenient, as a single sample from the augmentation distribution can provide an unbiased estimate $\hat{L}_{\text{loss}} = \log p\left(y_i | g(\mathbf{x}'_i; \theta)\right)$. Importantly though, a valid likelihood should arise from a valid distribution over labels, and should therefore normalize if we sum over labels. For instance, without augmentation,

$$\sum_{y_i=1}^{Y} \exp\left(L^i_{\text{noaug}}(y_i; \theta)\right) = 1. \tag{3.8}$$

However, if we try to interpret $L^i_{\text{loss}}\left(y_i; \theta\right)$ as a log-likelihood we find that it does not normalize to 1,

$$\sum_{y_i=1}^{Y} \exp\left(L^i_{\text{loss}}\left(y_i; \theta\right)\right) \neq 1. \tag{3.9}$$

and therefore $L^i_{\text{loss}}\left(y_i; \theta\right)$ cannot be the log of a valid probability distribution. Note that we might try to get a valid likelihood by including a normalizer. The problem is that this normalizer would need to depend on $\theta$, and thus would need to be included in the log-likelihood, and of course no normalizer terms appear in the loss (3.7). While we could renormalize $L^i_{\text{loss}} / \text{LogSumExp}_{\mathcal{Y}}\left(L^i_{\text{loss}}\right)$ to ensure validity, we will see in the next section that the form of $L^i_{\text{loss}}$ constitutes an unnecessarily slack bound on a principled log-likelihood, which we can tighten significantly.

### 3.2.2 Tighter Lower Bounds on the Log-likelihood of Principled DA Models

To incorporate DA into BNN likelihoods, we define the probabilities for each class to depend on averages over augmentations. We can choose to either average logits (equal to the NN outputs, $\mathbf{f}(\cdot; \theta)$) or predictive probabilities ($\text{softmax}\, \mathbf{f}(\cdot; \theta)$),

$$\mathbf{p}_{\text{inv}}(\mathbf{x}_i; \theta) = \mathbb{E}\left[\text{softmax}\, \mathbf{f}(\mathbf{x}'_i; \theta)\right], \tag{3.10}$$

$$\mathbf{f}_{\text{inv}}(\mathbf{x}_i; \theta) = \mathbb{E}\left[\mathbf{f}(\mathbf{x}'_i; \theta)\right]. \tag{3.11}$$

where we take expectations over $p(\mathbf{x}'_i|\mathbf{x}_i)$. Remember that $\mathbf{f}(\mathbf{x}'_i;\theta)$ is the (vector-valued) NN output for an augmented input, which is used as the logits in classification, so $\mathbf{f}_{\text{inv}}(\mathbf{x}_i;\theta)$ is the outputs averaged over all augmentations of the same underlying image. Likewise, $\mathbf{p}_{\text{inv}}(\mathbf{x}_i;\theta)$ is the vector of probabilities given by averaging the predicted probabilities over augmentations. These are denoted "inv" for invariant, because averaging over augmentations makes $\mathbf{f}_{\text{inv}}(\mathbf{x}_i;\theta)$ and $\mathbf{p}_{\text{inv}}(\mathbf{x}_i;\theta)$ invariant to $p\left(\mathbf{x}'_i|\mathbf{x}_i\right)$, which may not be true of the underlying NN, $\mathbf{f}(\mathbf{x}_i;\theta)$. The resulting (usually intractable) log-likelihoods are

$$\begin{aligned} L^i_{\text{prob}}\left(y_i;\theta\right) &= \log p_{\text{prob}}\left(y_i|\mathbf{x}_i,\theta\right) \\ &= \log \mathbb{E}\left[\text{softmax}_{y_i}\,\mathbf{f}(\mathbf{x}'_i;\theta)\right], \qquad (3.12) \\ L^i_{\text{logits}}\left(y_i;\theta\right) &= \log p_{\text{logits}}\left(y_i|\mathbf{x}_i,\theta\right) \\ &= \log \text{softmax}_{y_i}\,\mathbb{E}\left[\mathbf{f}(\mathbf{x}'_i;\theta)\right]. \qquad (3.13) \end{aligned}$$

These likelihoods were originally proposed in Wenzel et al. [2020] for averaging probabilities and [Van der Wilk et al., 2018] for averaging logits. However, they are computationally infeasible, as it is not (usually) possible to evaluate the expectation under all data augmentations. Instead, we need to choose an estimator or bound on these quantities. Wenzel et al. [2020] suggested a loose single sample bound for averaging probabilities, and Van der Wilk et al. [2018] suggested an unbiased estimator that is restricted to quadratic log-likelihoods. In contrast, we show that we can get tight, intuitive and easy to evaluate, multi-sample bounds analogous to those in Burda et al. [2015],

$$\begin{aligned} \hat{L}^i_{\text{prob},K}\left(y_i;\theta\right) &= \log\left(\frac{1}{K}\sum_{k=1}^{K}\text{softmax}_{y_i}\,\mathbf{f}(\mathbf{x}'_{i;k};\theta)\right), \\ \hat{L}^i_{\text{logits},K}\left(y_i;\theta\right) &= \log\text{softmax}_{y_i}\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{f}(\mathbf{x}'_{i;k};\theta)\right). \qquad (3.14) \end{aligned}$$

To prove the lower bound for averaging probabilities, we first rewrite the expectation inside the logarithm of (3.12) as the expectation of its average, over $K$ identically distributed random variables, $\mathbf{x}'_{i;k}$. We then take an approach familiar from variational inference [VI

Jordan et al., 1999] by applying Jensen's inequality to the (concave) logarithm function.

$$\begin{aligned} L^i_{\text{prob}}\left(y_i;\theta\right) &= \log \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \text{softmax}_{y_i}\,\mathbf{f}(\mathbf{x}'_{i;k};\theta)\right] \\ &\geq \mathbb{E}\left[\log \frac{1}{K}\sum_{k=1}^{K} \text{softmax}_{y_i}\,\mathbf{f}(\mathbf{x}'_{i;k};\theta)\right] \\ &= \mathbb{E}\left[\hat{L}^i_{\text{prob},K}\left(y_i;\theta\right)\right]. \end{aligned} \tag{3.15}$$

For averaging logits, we follow a similar method, noting that $\log \text{softmax}_{y_i}$ is a concave function [Boyd et al., 2004] taking a vector of logits and returning a scalar log-probability for class $y_i$. As such, we can again apply Jensen's inequality,

$$\begin{aligned} L^i_{\text{logits}}\left(y_i;\theta\right) &= \log \text{softmax}_{y_i}\,\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \mathbf{f}(\mathbf{x}'_{i;k};\theta)\right] \\ &\geq \mathbb{E}\left[\log \text{softmax}_{y_i}\,\frac{1}{K}\sum_{k=1}^{K} \mathbf{f}(\mathbf{x}'_{i;k};\theta)\right] \\ &= \mathbb{E}\left[\hat{L}^i_{\text{logits},K}\left(y_i;\theta\right)\right]. \end{aligned} \tag{3.16}$$

Finally, note that these objectives naturally correspond to the notions of averaging logits or averaging probabilities, which could be motivated using non-probabilistic considerations. Importantly, we do not claim the notion of averaging probabilities or averaging logits for different augmentations as a contribution in itself. We only claim as a contribution the notion that averaging probabilities or logits provide lower-bounds on principled log-likelihoods including DA, implying they can be used in a principled Bayesian setting, despite having some degree of stochasticity.

Increasing $K$ reduces the variance and tightens the bounds which eventually become

exact as $K \to \infty$ [Burda et al., 2015],

$$\mathbb{E}\left[\hat{L}^i_{\text{logits},K}\left(y_i;\theta\right)\right] \leq \mathbb{E}\left[\hat{L}^i_{\text{logits},K+1}\left(y_i;\theta\right)\right] \tag{3.17}$$

$$L^i_{\text{logits}}\left(y_i;\theta\right) = \lim_{K\to\infty} \hat{L}^i_{\text{logits},K}\left(y_i;\theta\right) \tag{3.18}$$

$$\mathbb{E}\left[\hat{L}^i_{\text{prob},K}\left(y_i;\theta\right)\right] \leq \mathbb{E}\left[\hat{L}^i_{\text{prob},K+1}\left(y_i;\theta\right)\right] \tag{3.19}$$

$$L^i_{\text{prob}}\left(y_i;\theta\right) = \lim_{K\to\infty} \hat{L}^i_{\text{prob},K}\left(y_i;\theta\right). \tag{3.20}$$

However, larger $K$ introduces greater computational cost. We therefore consider what value of $K$ is sensible by plotting the bound against $K$. We indeed found that the bound increases with $K$ up to around 10, when it saturates (Figure 3.1). While these differences might seem small when evaluated purely at test-time, they seem to cause much larger differences when integrated into training (Figs. A.1 and 3.2).

In contrast to good performance requiring $K \sim 10$, VI practitioners frequently use a single-sample bound. However, VI the expectation is taken over the approximate posterior distribution, which is optimised during inference. This significantly reduces the variance of the Monte Carlo estimator. In principle, similar variance reduction strategies exist in our setting, but would involve learning a separate variance-reducing augmentation distribution for each image, which is clearly impractical.

We note that our derived bounds with $K = 1$ represent such crude approximations that the differences between averaging probabilities, logits and losses collapse,

$$\hat{L}^i_{\text{logits};1}\left(y_i;\theta\right) = \hat{L}^i_{\text{prob};1}\left(y_i;\theta\right) = \hat{L}^i_{\text{loss}}\left(y_i;\theta\right) \tag{3.21}$$

which are all equal to $\log \text{softmax}_{y_i} \mathbf{f}(\mathbf{x}'_{i;1};\theta)$. We emphasise that the slackest bound in our family of estimators ($K = 1$) is the commonly used DA log-likelihood $\hat{L}^i_{\text{loss}}$, suggesting performance may benefit from $K > 1$.

### 3.2.3 Finite Orbit

Finally, all of the above is for the usual "full orbit" setting, where there is a distribution over a very large, or even infinite number of possible augmentations. The full orbit setting necessitates the use of the bound in (3.14), and allows us to use different numbers of samples at test and training time, $K_{\text{test}}$ and $K_{\text{train}}$ respectively. Remarkably, if we consider an alternative "finite orbit" by restricting the augmentations to a small subset, we can

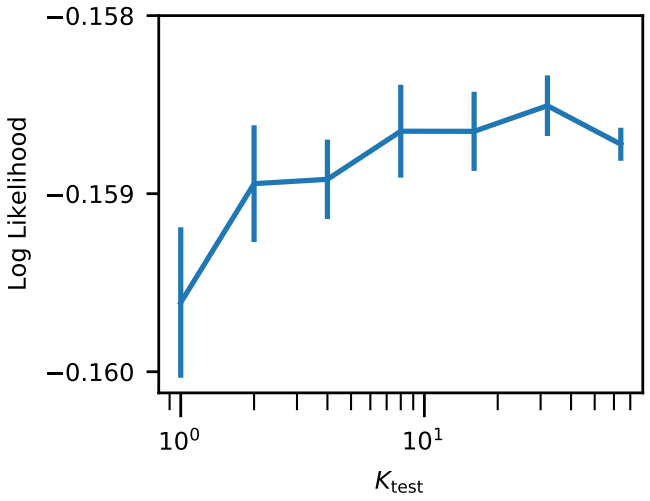



Figure 3.1: **The effect of $K_{\text{test}}$ on the log-likelihood bound.** We use a test batch (size 512) from CIFAR-10. Values shown for ResNet20 BNN trained and tested with $\hat{L}_{\text{prob},K}$ ($K_{\text{train}} = 8$ and $T = 0.001$). Error bars cover two standard errors above/below mean for DA sampling with different seeds. Sixty seeds used for $K_{\text{test}} = \{1, 2\}$, thirty for $K_{\text{test}} = 4$ and five for all other $K_{\text{test}}$.

*exactly* evaluate the log-likelihood. In the finite orbit setting, the conditional augmentation distribution can be written as

$$p\left(\mathbf{x}_i' \middle| \mathbf{x}_i\right) = \frac{1}{K} \sum_{k=1}^{K} \delta\left(\mathbf{x}_i' - a_k(\mathbf{x}_i)\right), \tag{3.22}$$

where $\delta$ is the Dirac-delta, and $a_k$ is a function that applies the $k$th fixed augmentation. In this setting, it is possible to exactly compute $L^i_{\text{logits}}(y_i; \theta)$ and $L^i_{\text{prob}}(y_i; \theta)$ by summing over the $K$ augmentations. This allows us to empirically explore how exact log-likelihood computation influences the CPE, comparing it with the bounds in the full-orbit setting (3.14) in Section 3.3.1. When implementing finite orbit augmentation in practice, we choose the $K$ fixed augmentations by sampling them before training. The finite orbit setting uses the same augmentations, and therefore the same number of augmentations, at test and train time: $K_{\text{train}} = K_{\text{test}} = K$.

### 3.2.4 Non-categorical Likelihoods

While we have derived the log-likelihood bounds for categorical likelihoods, averaging either logits (3.16) or probabilities (3.15), we emphasise that analogous bounds exist for other likelihoods. As above, we denote the NN as $\mathbf{f}(\cdot;\theta)$, but note its output could now parameterise an arbitrary likelihood function.

Averaging probabilities can be immediately generalised,

$$p_{\text{prob}}(\mathbf{y}_i|\mathbf{x}_i,\theta) = \int p\left(\mathbf{y}_i|\mathbf{f}\left(\mathbf{x}'_i;\theta\right)\right) p\left(\mathbf{x}'_i|\mathbf{x}_i\right) \mathrm{d}\mathbf{x}'_i = \mathbb{E}\left[p\left(\mathbf{y}_i|\mathbf{f}\left(\mathbf{x}'_i;\theta\right)\right)\right], \tag{3.23}$$

giving the $K$-sample bound

$$\hat{L}^i_{\text{prob},K}(\mathbf{y}_i;\theta) = \log\left(\frac{1}{K}\sum_{k=1}^{K} p\left(\mathbf{y}_i|\mathbf{f}\left(\mathbf{x}'_{i,k};\theta\right)\right)\right), \quad \mathbf{x}'_{i,k} \sim p\left(\mathbf{x}'_i\middle|\mathbf{x}_i\right). \tag{3.24}$$

Intuitively, each augmentation in (3.23) forms one component of a (potentially infinite) mixture model over the outputs, $\mathbf{y}_i$.

To generalise the logit-averaging bound, we directly average the parameters of the distribution predicted by the latent NN function,

$$p_{\text{latent}}(\mathbf{y}_i|\mathbf{x}_i,\theta) = \int p\left(\mathbf{y}_i|\bar{\mathbf{f}}(\mathbf{x}_i;\theta)\right), \tag{3.25}$$

$$\bar{\mathbf{f}}(\mathbf{x}_i) := \int \mathbf{f}\left(\mathbf{x}'_i\right) p\left(\mathbf{x}'_i|\mathbf{x}_i\right) \mathrm{d}\mathbf{x}'_i \tag{3.26}$$

giving the $K$-sample bound

$$\hat{L}^i_{\text{latent}}(\mathbf{y}_i;\theta) = \log\left(p\left(\mathbf{y}_i\middle|\frac{1}{K}\sum_{k=1}^{K}\mathbf{f}\left(\mathbf{x}'_{i,k};\theta\right)\right)\right), \quad \mathbf{x}'_{i,k} \sim p\left(\mathbf{x}'_i\middle|\mathbf{x}_i\right). \tag{3.27}$$

Note that in this case we require that $p\left(\mathbf{y}_i\middle|\mathbf{f}\left(\mathbf{x}'_{i,k};\theta\right)\right)$ is concave wrt $\mathbf{f}\left(\mathbf{x}'_{i,k};\theta\right)$ (for fixed $\mathbf{y}_i$) in order for (3.27) to be a lower bound on the log-likelihood.

## 3.3 Experiments

### 3.3.1 Setup

Next we ask: how is the CPE influenced when DA is incorporated into the model in a principled way? We take the code[2] and networks from Fortuin et al. [2022, 2021] and mirror their experimental setup for CIFAR-10 and MNIST as closely as possible. This code combines a cyclical learning rate schedule [Zhang et al., 2019], an SGLD sampling scheme [Garriga-Alonso and Fortuin, 2021], and the preconditioning and convergence diagnostics from Wenzel et al. [2020]. We use the CIFAR-10 DA transformations from Wenzel et al. [2020], Fortuin et al. [2022], Noci et al. [2021]: 1. random cropping with a padding of four pixels and 2. horizontal flipping, with probability 0.5. For MNIST we apply 1. random cropping with a padding of two pixels, then 2. random rotation by an angle sampled uniformly over $(-\pi/6, \pi/6)$. Following Fortuin et al. [2022], we run 60 cycles with 50 epochs per cycle. We record one sample at the end of each of the last five epochs of a cycle, giving 300 samples total. For architectures, we use a ResNet20 for CIFAR-10 as in Wenzel et al. [2020] and Fortuin et al. [2022]. For MNIST, we use the three-layer fully connected network (FCNN) from Fortuin et al. [2022]. The experiments took around 90 GPU-days on Nvidia RTX6000s[3].

**Evaluation** We evaluate the performance of the Monte Carlo-estimated Bayesian model average (1.20),

$$p\left(\mathbf{y}^{*}|X^{*}\right) \approx \frac{1}{N_{\mathrm{MC}}} \sum_{j=1}^{N_{\mathrm{MC}}} \prod_{n=1}^{N^{*}} p\left(y_{n}^{*}|g_{K}\left(\mathbf{x}_{n}^{*}, \theta_{j}\right)\right), \quad \theta_{j} \sim p_{T}\left(\theta|\mathbf{y}, X\right) \tag{3.28}$$

where $(X^{*}, \mathbf{y}^{*})$ is the test data comprising $N^{*}$ examples, and $g_{K}\left(\cdot; \theta\right)$ is either i) the NN (no test-time DA), ii) the predicted probabilities, averaged over $K$ test-time augmentation samples, iii) the predicted logits, averaged over $K$ test-time augmentation samples, as labelled in the results. We evaluate the predictions according to i) the classification accuracy, ii) the log-likelihood and iii) the expected calibration error [ECE; Naeini et al., 2015].

[2] github.com/ratschlab/bnn_priors; MIT Licensed
[3] Code available: github.com/sethnabarro/bnn-data-aug/

### 3.3.2 Results

The results are presented in Figure 3.2. We replicate the finding that the CPE is largely absent without DA (dashed black line), and is present in the standard setup with DA at training time ($K_{\text{train}} = 1$) but without augmentation at test time (solid black). Further, we show that the CPE persists with principled DA likelihoods: averaging logits with full orbit (purple, first and third rows), and averaging probabilities with finite and full orbits (green).

For CIFAR-10, the best method overall appears to be averaging probabilities with a full orbit (dark green line, third row) at $T = 10^{-3}$, though at $T = 1$ averaging logits (dark purple lines) outperforms the other methods. For the MNIST experiments, logit averaging over a full orbit (purple line, top row) performs best at all temperatures, though has a similar accuracy to averaging probabilities (green line, top row) at $T = 10^{-3}$. Interestingly, the CPE for averaging probabilities (green) is stronger than that for both logit averaging (dark purple) and standard DA (solid black), across all MNIST experiments. We highlight that our logit-averaging, full orbit bound is more accurate than standard DA for both datasets, at all temperatures.

For both datasets, the CPE is near absent in one particular setting: averaging logits with a finite orbit (purple line, second and fourth rows). However, the relevance of this is unclear, as for CIFAR-10 it is clearly the worst performing of all DA approaches, and for MNIST it is outperformed by standard DA. Indeed, remember that the arguments for the optimality of Bayesian inference apply only in the case that the model is well-specified [Kolmogorov, 1950, Savage, 1954, Jaynes, 2003]. However, the comparatively poor performance of averaging logits with a finite orbit indicates that it is likely to be the wrong model, while other settings are likely to be closer to the true model. In that case, the presence or absence of the CPE in the wrong model (averaging logits with a finite orbit) is immaterial to our understanding of the CPE in the right model. Note that this argument could not be made if there was a model without the CPE with performance equal to or better than the other models (see Section 3.6 for further discussion).

The CPE was originally discovered in Wenzel et al. [2020] when assessing test accuracy and log-likelihood — they did not consider other measures of distribution calibration like expected calibration error (ECE). Indeed, later work on the CPE found that measures such as ECE are far more complex and usually do not agree with test accuracy and log-likelihood [Fortuin et al., 2021]. It is therefore difficult to interpret the differences between test log-likelihod and ECE, especially if we remember that test log-likelihood is itself a proper

scoring rule [Gneiting and Raftery, 2007], and therefore captures one possible notion of calibration. In particular, test log-likelihood heavily penalizes an event assessed as low probability actually happening, e.g. if our classifier predicts a probability of 0.001%, while the actually happens even 0.1% of the time. In contrast, ECE considers the absolute difference in probability, so it far more heavily penalises e.g. a predicted probability of 40% while the event actually happens 60% of the time. Needless to say, the most appropriate measure of calibration will depend heavily on the domain, with log-likelihood being more appropriate for low-probability but high risk events. In our CIFAR-10 experiments, averaging probabilities (green) achieves the greatest log-likelihood scores, standard DA (solid black) achieves the lowest ECE. This is contrasted with MNIST, for which averaging logits (purple) has highest log-likelihood and no DA (dashed black) has lowest ECE.

The usefulness of our results is contingent on understanding whether we are indeed accurately approximating the posterior. To check this, we computed the kinetic temperature [Leimkuhler and Matthews, 2015], which estimates the temperature of a given parameter in the Langevin dynamics simulation from the norm of its momentum. In expectation, the kinetic temperature estimator should be equal to the desired temperature, $T$. The results (Appendix A.2) indicate that all the samplers run at their desired temperature, a result that is consistent with accurate posterior sampling.

As discussed in Section 3.2, increasing $K$ tightens our log-likelihood bounds, but incurs greater computational cost. It is natural to question which value of $K$ is a good trade-off. We explore how the log-likelihood of test data under a trained model varies with $K_{\text{test}}$. As expected, the results (Figure 3.1) show the log-likelihood increases with $K_{\text{test}}$, with even $K = 2$ being a significant improvement over $K = 1$ (standard DA). However, the curve plateaus, suggesting that for CIFAR-10, there is little benefit of using $K > 8$.

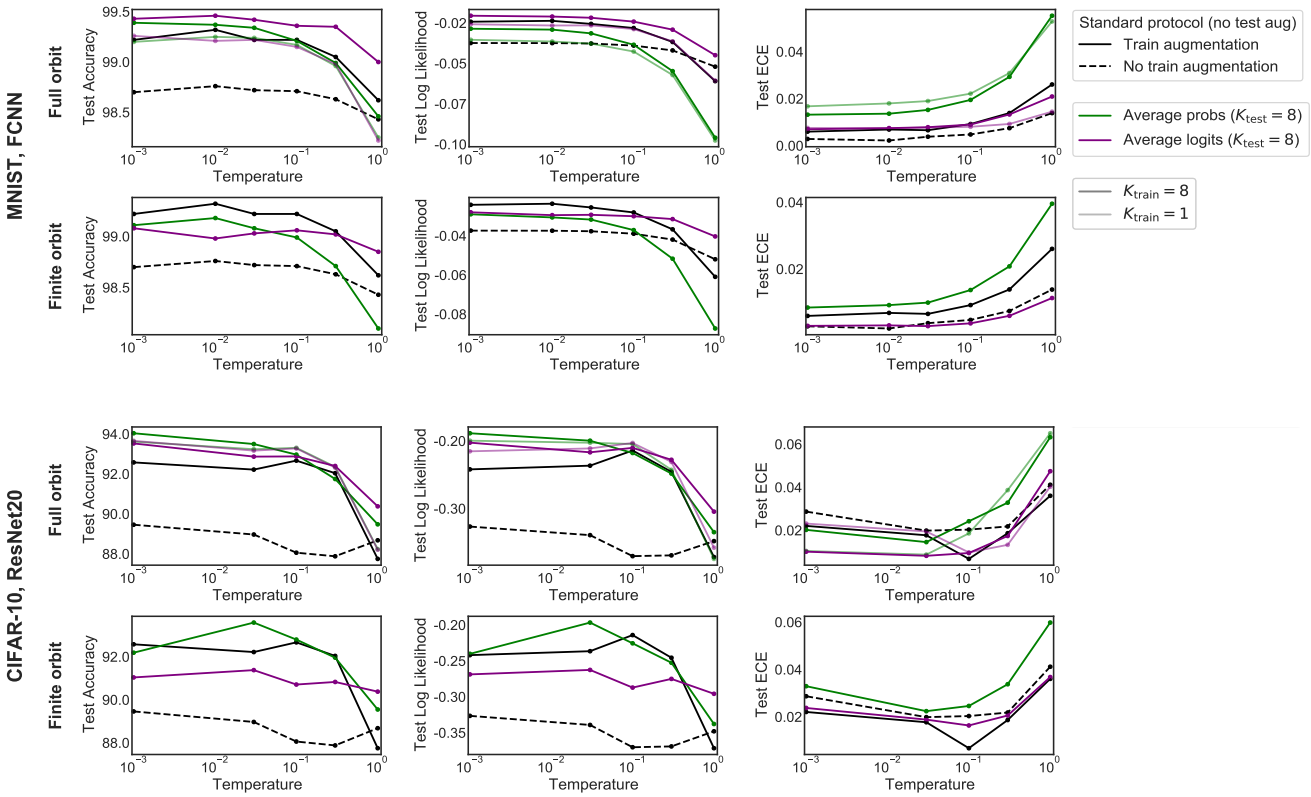



Figure 3.2: **The effect of different DA configurations on the CPE.** In all settings we do Bayesian inference using SGLD as per Garriga-Alonso and Fortuin [2021]. Without DA, there is a minimal CPE. Most other configurations show significant improvement for $T < 1$, with the exception of averaging the logits over a finite orbit. Averages computed with $K_{\text{train}} = 8$ and $K_{\text{test}} = 8$.

## 3.4 Two perspectives on Bayesian data augmentation

As discussed in Section 3.2.4, the lower bounds with logit-averaging and probability-averaging are specific instances of invariant likelihood bounds, which can be applied to arbitrary likelihood functions. We now explore the implications of these two approaches to probabilistic DA, (3.23) and (3.25), in more general terms, not in the specific context of BNNs and the CPE. We discuss their motivations in Section 3.4.1 and 3.4.2, and compare their properties in Section 3.4.3.

### 3.4.1 Invariance Construction

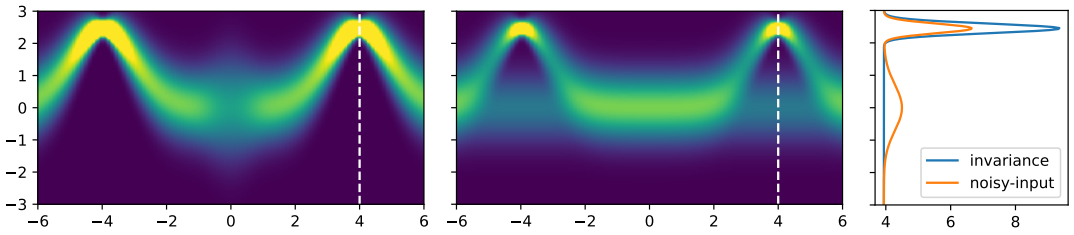
When averaging the latent NN output, or logits (3.27), we aim to make the NN mapping $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^C$ invariant to DA transformations by averaging the outputs. This construction influences only the regression function, and so has a similar effect to changing the NN architecture or changing the prior on the functions $\mathbf{f}(\cdot)$ in the Bayesian case [van der Wilk et al., 2018]. Since only the outputs are affected, this can be directly applied to any concave likelihood that depends only on an evaluation of the function, i.e. any concave likelihood which can be written as $p(\mathbf{y}_i|\mathbf{f}_i)$. We refer to this method as the *invariance construction*.

### 3.4.2 Noisy Input Model

This probability-averaging method (3.24), was also discussed in Appendix K of Wenzel et al. [2020] and is a (potentially continuous) mixture model on the observation $\mathbf{y}_i$, where each augmentation introduces a mixture component. This is as a *noisy input* model [Girard and Murray-Smith, 2003, McHutchon and Rasmussen, 2011, Damianou et al., 2016] where the input $\mathbf{x}_i$ is corrupted via the augmentation distribution.

### 3.4.3 Model Comparison

The forms of the invariance construction (3.25) and the noisy input model (3.23) imply a difference of purpose. In using the invariance construction, we seek a regression function with the specified symmetry, which is consistent with the data according to the likelihood function $p\left(\mathbf{y}_i|\mathbf{f}_i\right)$. Conversely, with the noisy input model (3.23) we aim to find a function which gives rise to an invariant likelihood, consistent with the observed outputs for inputs randomly perturbed by $p(\mathbf{x}'|\mathbf{x})$. The role of $\mathbf{x}'$ is different in each case. In the noisy input model, $\mathbf{x}'$ is a latent variable on which we could, in principle, do inference (with e.g. an




(a) Invariant GP (b) noisy input GP (c) $p(y|x=4,\mathcal{D})$

Figure 3.3: **Invariant and noisy-input GP posterior densities.** Both models are conditioned on a single observation at $x_1 = -4, y_1 = 2.5$.

amortized variational approach). While in the invariance construction, we integrate over $\mathbf{x}'$ to parameterise $\mathbf{f}(\mathbf{x};\theta)$.

We now compare the behaviours of the invariance and noisy input constructions. We will see that they result in quite different posteriors.

In Section 3.3, we compared the empirical performance of averaging probabilities and averaging logits for BNN classification (see Figure 3.2). In this case, the invariance perspective could justify both averaging the logit vector or the corresponding probability vector — both are sufficient parameterisations of a categorical distribution — and so classification does not cleanly distinguish between the noisy input and invariance constructions. With this in mind, we instead investigate the models with an illustrative regression example, which enables both integration over the orbit and posterior inference in closed form.

We consider Gaussian process (GP) regression with a one-dimensional input and data augmentation which enforces symmetry about $x = 0$, i.e. $p(x'|x) = \frac{1}{2}(\delta(x' - x) + \delta(x' + x))$. From Van der Wilk et al. [2018], the invariance view can be expressed in the kernel of the GP:

$$g \sim \mathcal{GP}(\mathbf{0}, k_{\text{base}}) \tag{3.29}$$

$$f(x) = g(x) + g(-x) \tag{3.30}$$

$$\implies f \sim \mathcal{GP}(\mathbf{0}, k_{\text{inv}}), \tag{3.31}$$

$$\text{where} \quad k_{\text{inv}}(x_i, x_j) = \sum_{c_i \in \{-1,1\}} \sum_{c_j \in \{-1,1\}} k_{\text{base}}(c_i x_i, c_j x_j). \tag{3.32}$$

We then follow standard GP inference to find the posterior over invariant functions. Note that unlike Van der Wilk et al. [2018], we are not concerned with learning invariances here.

The noisy input model for this case is

$$p(\mathbf{x}, \mathbf{y}, \mathbf{f}) = p(\mathbf{f}) \prod_{i=1}^{N} \int p\left(y_i | f(x_i')\right) p\left(x_i' | x_i\right) \mathrm{d}x_i' \tag{3.33}$$

$$p(y_i | f(x_i')) = \mathcal{N}\left(y_i; f(x_i'), \sigma^2\right) \tag{3.34}$$

$$f \sim \mathcal{GP}(0, k). \tag{3.35}$$

Given a single observation $(x_1, y_1)$, the noisy input posterior is

$$p\left(f | x_1, y_1\right) = \frac{1}{Z} p(f(x_1), x_1, y_1) \tag{3.36}$$

$$= \frac{1}{2Z} p\left(f(x_1)\right) \left[p\left(y_1 | f(x_1)\right) + p\left(y_1 | f(-x_1)\right)\right] \tag{3.37}$$

$$= \frac{1}{2} \left[p\left(f | x_1, y_1\right) + p\left(f | -x_1, y_1\right)\right], \tag{3.38}$$

a mixture of GP posteriors, with two components (one for each point in the orbit).

How do these posteriors compare? For an observation at $(x_1 = -4, y_1 = 2.5)$ we plot the posterior predictive densities in Figure 3.3. Both posteriors are symmetric around $x = 0$ as we expect, however the noisy input model is bimodal in the regions surrounding $x = 4$ and $x = -4$, where the invariance posterior has unimodal density concentrated around the observed $y$ value of 2.5. The difference is clear in Figure 3.3c, which shows the marginal predictive densities at $x = 4$.

In the noisy input case, our observation is $(x_1, y_1)$, but $x$ is uncertain, so the observation could have been generated by $(-x_1, y_1)$ with equal probability. This results in a mixture posterior with two components: one component has "seen" $(x_1, y_1)$, while the other "saw" $(-x_1, y_1)$. The first component's prediction at $-x_1$ remains uninformed by its "observation" and the same is true for the second component's prediction at $x_1$. Thus, the predictions made by these components at these locations revert to the zero-mean prior.

From the invariance perspective, we condition on the point $(x_1, y_1)$ but the double-sum kernel forces the function to be the same at $(-x_1, y_1)$. As the posterior is a single GP, it has unimodal marginals with high density around both points.

We can gain further intuition by looking at samples from both posteriors (Figure 3.4). We can see that the *every* sample from the invariance posterior is symmetric about $x = 0$,

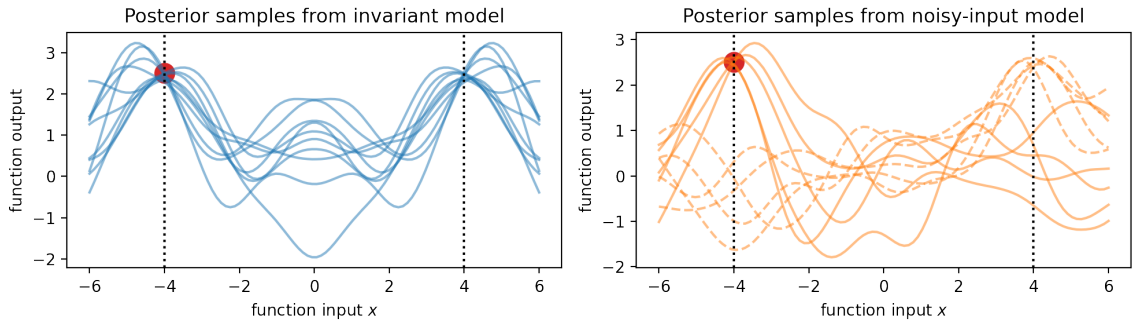
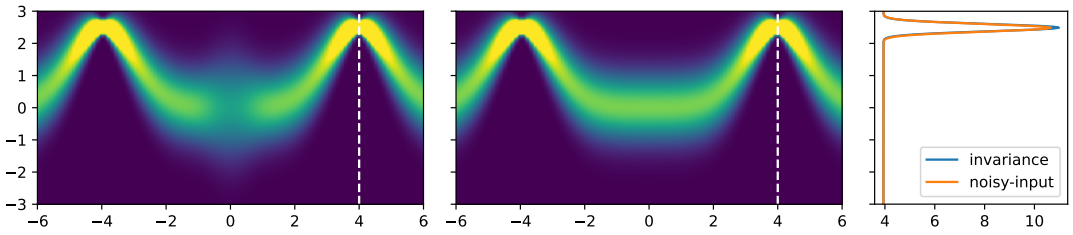



Figure 3.4: **Invariant and noisy-input GP posterior samples.** Both models are conditioned on a single observation at $x_1 = -4, y_1 = 2.5$. The noisy input posterior comprises two components: one conditioned on $(x_1, y_1)$ (dashed lines), the other on $(-x_1, y_1)$ (solid lines).

where the functions drawn from the noisy input posterior are not symmetric in general.

The samples illustrate the key difference between the models. For the noisy input model, we can see the two components of the mixture posterior arise from conditioning on different locations in the orbit of $x_1$ as described above. The component going through $(x = 4, y_1)$ (samples drawn with dashed lines) is close to the prior at $(x = -4, y_1)$, the other (solid lines) goes through $(x = -4, y_1)$ and is close to the prior at $(x = 4, y_1)$. However, under the invariance model, inference on the observation concentrates all model density around $y_1$ for both points in the orbit of $x_1$.

We now consider how this comparison changes as we observe more data. The noisy input model (3.33) requires integration over $p(x'|x)$ to compute the likelihood of each datapoint, all of which are multiplied together to calculate their combined likelihood. Thus, the number of posterior components grows exponentially with the number of observations: $A^N$ (for orbit size $A$). Suppose all observations are at the same location $(x_1, y_1)$. In this case, the posterior density due to prior reversion at $\{x_1, -x_1\}$ decreases exponentially with $N$. This is because the fraction of mixture components conditioned on all input observations being at the same point in the orbit of $x_1$, i.e. all at $x_1$ or $-x_1$, is given by $A^{1-N}$. The predictive posteriors for ten observations, each at $(x = -4, y = 2.5)$ is shown in Figure 3.5. Contrasting this noisy input posterior (Figure 3.5b) to that for one observation (Figure 3.3b), we can see the reduction in density around to the prior mean for points around the orbit of $x_1$. In summary, the noisy input and invariance posteriors become more alike as we observe more data in the same orbit.




(a) Invariant GP (b) noisy input GP (c) $p(y|x=4,\mathcal{D})$

Figure 3.5: **Invariant and noisy-input GP posterior densities.** Both models are conditioned on ten observations at $x_1 = -4, y_1 = 2.5$.

## 3.5 Related Work

Past work introduced noisy input generative models which average probabilities [Wenzel et al., 2020]. However, this work did not consider the tighter multi-sample bounds developed here, or the finite orbit setting which allows us to evaluate the exact likelihood. This left open the possibility raised by Izmailov et al. [2021] that the CPE was an artefact of standard DA resulting in an invalid likelihood. In contrast, we considered exact likelihoods in the finite orbit setting, and tighter multi-sample lower bounds in the full orbit setting. Further, the invariant function perspective allowed us to derive a log-likelihood bound for averaging logits, not considered by Wenzel et al. [2020]. As the CPE persists when using our principled DA models, we can exclude the possibility that the CPE is an artefact of DA giving a "randomly perturbed log-likelihood". Other work has introduced a log-likelihood estimator for averaging GP logits using the invariance principle [van der Wilk et al., 2018]. However, the method only works for a quadratic log-likelihood and thus necessitates Pólya-Gamma approximations for classification. Further, the work did not consider BNNs or the connection to the CPE.

There is a small but growing body of work that considers averaging over multiple augmentations at training time [Hoffer et al., 2019, Berman et al., 2019, Choi et al., 2019, Benton et al., 2020, Lyle et al., 2020, Touvron et al., 2021, Fort et al., 2021]. However, this work was not done within a Bayesian framework (e.g. by using SGLD or a similar inference algorithm), did not show that averaging across multiple training augmentations gives a multi-sample bound on the log-likelihood of a principled model, did not consider the finite-orbit setting where the log-likelihood can be computed exactly, and did not consider the interaction with the CPE. In addition, much of this work uses averaging losses [Hoffer

et al., 2019, Berman et al., 2019, Choi et al., 2019, Benton et al., 2020, Touvron et al., 2021, Fort et al., 2021] which is equivalent to using a loose single-sample bound on the log-likelihoods. While Lyle et al. [2020] show that feature averaging during training can improve generalization, our work is, to the best of our knowledge, the first to average predicted probabilities at training time. Finally, the idea of averaging at test-time is more common and has been practiced for longer [e.g. Krizhevsky et al., 2012, Simonyan and Zisserman, 2014, He et al., 2015, Szegedy et al., 2015, Foster et al., 2020].

A considerable body of past work on BNNs uses DA, both with VI [Blundell et al., 2015, Zhang et al., 2018, Osawa et al., 2019, Ober and Aitchison, 2020, Unlu and Aitchison, 2021], Laplace approximations [Immer et al., 2021] and SGLD [e.g. Zhang et al., 2019, Fortuin et al., 2022, Wang and Aitchison, 2021]. However, as discussed in Section 3.2.1, these methods simply substitute non-augmented for augmented data and thus do not use a valid log-likelihood. In contrast, we incorporated DA into the probabilistic generative model, and thus are able to give valid log-likelihoods based on averaging logits or averaging probabilities in the classification case.

## 3.6 Conclusion

We have shown how DA can be properly incorporated into a generative model suitable for BNN inference by deriving a lower-bound on the log-likelihood of the augmentation-averaged network output. We have discussed these in the general context of Bayesian DA in Section 3.4. More specific to BNNs, our empirical analysis (Section 3.3) has shown that i) the CPE persists even when using our principled DA formulation, and ii) in agreement with past work [Wenzel et al., 2020, Fortuin et al., 2022, Izmailov et al., 2021], we see the CPE is absent without DA.

What do these results imply for the origin of the CPE? Our models in principle have a clean log-likelihood which can be evaluated exactly in the finite orbit setting, or which we estimate using tightened multi-sample bounds in the full orbit setting. This falsifies the hypothesis, that the CPE is an artefact arising from DA giving a "randomly perturbed log-likelihood [which] does not have a clean interpretation as a valid likelihood function".

Indeed, it is worth stepping back and considering the original motivation for studying the CPE, namely that if we have the correct model, then Bayesian inference with $T = 1$ should give optimal performance [Kolmogorov, 1950, Savage, 1954, Jaynes, 2003, Wenzel et al., 2020]. Critically, we need the right model for us to expect optimal performance at

$T = 1$. We now have two classes of model, with DA and without DA, so which is right(er)? Given the significant and widely recognised performance benefits of DA, it seems very likely that the "right" model would include some form of DA. If the model with DA is right(er), and that model displays the CPE, then the CPE still demands an explanation, and the presence or absence of the CPE in the wrong model without DA is immaterial. As such, the presence of the CPE in models with DA remains an important problem, and is likely to be caused by one of the two other explanations discussed in Section 3.1: either data curation [Aitchison, 2020] or prior misspecification [Wenzel et al., 2020, Fortuin et al., 2022]. Indeed, we would tentatively suggest the opposite of Izmailov et al. [2021]: that it is in reality the *lack* of a CPE without DA that is an artefact of using the wrong model (i.e. without DA).

# Chapter 4

# Learning in Deep Factor Graphs with Gaussian Belief Propagation

In this chapter, we cover the work from our publication *Learning in Deep Factor Graphs with Gaussian Belief propagation*, presented at the *International Conference on Machine Learning 2024* [Nabarro et al., 2024].

Our main contribution in this chapter is *GBP Learning*, a framework for learning in NN-inspired factor graphs with Gaussian belief propagation (GBP). We treat all relevant quantities (inputs, outputs, parameters, activations) as random variables in a graphical model, and view training and prediction as inference problems with different observed nodes. Our experiments show that these problems can be efficiently solved with GBP, whose updates are inherently local, presenting exciting opportunities for distributed and asynchronous training. Our approach can be scaled to deep networks and provides a natural means to do continual learning: use the estimated posterior of the current task as a prior for the next.

Experimentally, we demonstrate the benefit of learnable parameters over a classical factor graph approach on a video denoising task and we show encouraging performance for continual image classification. In the latter context, we find GBP Learning can perform well with randomly ordered message updates and in model-parallel regimes with infrequent inter-processor communication. We also present an experiment demonstrating how our method can combine learnable and hand-designed models. We finish with conclusions and directions for future work.

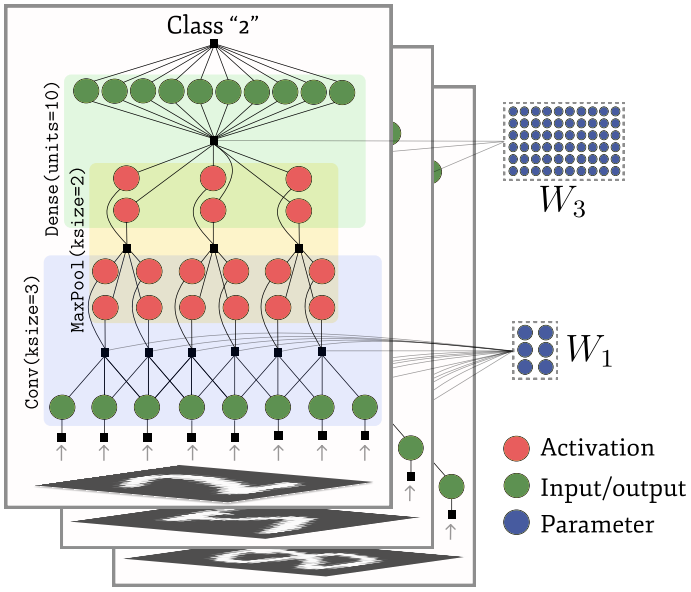



Figure 4.1: In **GBP Learning**, we design factor graphs whose structure mirrors common NN architectures, enabling distributed training and prediction with GBP. Learnable parameters are included as random variables (circles), as are inputs, outputs and activations. The parameters are shared over across all observations, where the other variables are copied once per observation. Factors (black squares) between layers constrain their representations to be locally consistent, while those attached to inputs and outputs encourage compatibility with observation. The inter-layer factors are non-linear to enable soft-switching behaviour. This example architecture for image classification comprises convolutional, max pooling and dense projection layers. The same architecture could be trained without supervision by removing the output observation factor.

## 4.1 Introduction

Deep learning (DL) has been transformative across many domains. However, its applicability is limited in cases where i) we require efficient, robust representations which can be trained incrementally; ii) supervision is sparse or irregular; and iii) learning must augment, or run alongside, hand-designed solvers. Pretrained models, when available, might mitigate these limitations, but struggle as train and test distributions diverge. How they should be updated online remains an open research question.

Concurrently, we reflect on how neural networks (NNs) are trained. Despite backpropagation [Rumelhart et al., 1985] being largely successful for DL, training NNs spread over

multiple processors is made difficult by *backward locking*: processors for earlier layers sit idle after their part in the forward pass, awaiting the backward error signal. This challenge will become more pertinent with the growth of i) larger models which must be distributed over many devices, ii) new hardware architectures whose cores have significant local memory [Graphcore, Cerebras], and iii) parallel, distributed and heterogeneous embedded devices [Sutter, 2011]. We thus expect a growing need for more flexible training algorithms which admit efficient model parallelism.

We argue the above challenges essentially relate to the fusion of multiple signals: old and new (for incremental learning); hand-crafted vs learnt; and representations across multiple layers of a model (distributed training). Bayesian principles offer a clear answer on how to fuse: signals should be combined according to the rules of probability. We seek to exploit this fact in our probabilistic approach to DL. Specifically, our models are factor graphs (Figure 4.1) with random variables for all quantities relevant to DL: inputs, outputs, activations and parameters. This representation enables continual learning via online updating of the parameter posterior, and interoperability through connections with other factor graphs. We seek to design the models so as to retain the properties which we believe make DL powerful. Namely random initialisation and over-parameterisation [Allen-Zhu et al., 2019]; architectural motifs encoding good inductive biases; and non-linearities which switch to selectively activate and prune when exposed to data [Glorot et al., 2011].

Much recent work has shown GBP to be a robust and effective algorithm for distributed inference in factor graphs, even in the presence of non-linear and non-Gaussian factors [Davison and Ortiz, 2019, Ortiz et al., 2020, Murai et al., 2023, Patwardhan et al., 2022]. It is thus our choice of inference engine here. By approximating the factors in our model as Gaussian, we can use GBP for training (inference over parameters, given observations) and prediction (inference of e.g. outputs, given parameters and inputs). The generality of GBP provides flexibility as to which variables are observed, meaning training and prediction are essentially the same computation, and partial observations or missing labels do not require fundamentally different treatment. As BP is inherently local and stateful, our training can be distributed and asynchronous.

Our work has similarities with that of Lucibello et al. [2022], which came to our attention after we developed GBP Learning. Lucibello et al. [2022] train MLP-like factor graphs using GBP with analytically derived message updates. For computational efficiency, they focus on architectures with binary weights and sign activation functions. In contrast, our approach enables training with GBP on arbitrary architectures, without rederivation of the message

updates; and we focus on models with continuous weights in natural analogue to NNs. We demonstrate this generality, training convolutional architectures with our approach and showing we can outperform Lucibello et al. [2022] on image classification tasks.

We call our approach *GBP Learning*. Our models (Section 4.2.1) are factor graphs with architectures inspired by those in DL, and factors between layers to enforce multi-layer consistency as used in predictive coding models [Millidge et al., 2022]. Within these models, GBP admits flexible and distributed training and prediction (Section 4.2.2), and learning can be done incrementally via Bayesian filtering over parameters (Section 4.2.4). Our experiments demonstrate the benefit of factor graphs with learnable parameters over hand-designed solvers in a video denoising task (Section 4.3.2.1). For image classification (Section 4.3.3), we evaluate our continual learning approach by single epoch training on MNIST. We achieve performance equivalent to an Adam-trained CNN with a replay buffer of $6 \times 10^3$ examples, and show this performance is robust to asynchronous and model-parallel training. We compare to Lucibello et al. [2022] (Section 4.3.3.5), outperforming them on both MNIST (by 0.8%) and CIFAR-10 (by 11.8%). We finish with a demonstrative example (Section 4.3.4), showing how our approach enables the combination of learnable and hand-designed factor graph models.

To summarise, our main contributions are as follows:

1. An **approach to train deep factor graphs with GBP**, which can be applied to any architecture and supports incremental learning.
2. **Experimental results** which show promise for asynchronous, model-parallel, continual image classification and video denoising.

## 4.2 GBP Learning

Our goal is to design factor graphs that share architectural benefits with NNs, such as their inductive biases and overparameterisation, while allowing efficient, decentralised training through GBP. We enable the learning of parameters with GBP by including them as random variables in the graph.

In addition to the computational properties, an advantage of training with GBP is that parameters can be identified online, using the same inference mechanisms employed for traditional state estimation. This contrasts with methods like expectation maximisation [Dempster et al., 1977] or contrastive divergence [Hinton, 2002], where parameter updates

and latent variable inference occur in separate phases. GBP Learning could therefore enable a unified process that *simultaneously* learns functions from data and estimates state variables.

The approach we describe below represents just one way to achieve our goal. While we have found it to work well in practice, there is a broad design space with many other potential architectures that could perform as well or better. Possible alternatives include: using vector-valued instead of scalar variables, incorporating multiple NN layers within each factor connecting activation variables, replacing nonlinear activations with robust energy factors, and experimenting with alternative GBP message schedules.

We will now describe the key factors in our model and our efficient GBP routine for training and prediction. The factor energies of form (2.71) contain measurement function $h(\cdot)$ and observations $\mathbf{y}$ which, along with $\Lambda_{\mathbf{y}}$, are sufficient to deduce the linearised factor parameters (2.76) and (2.77). This, in turn, defines the GBP message updates (2.66), (2.67). It is thus sufficient to describe our model in terms of factor energies. We emphasise that our model parameters are included as random variables in the factor graph and inferred with GBP.

### 4.2.1 Deep Factor Graphs

We design networks to find representations based on local consistency, i.e. the activations $\mathbf{h}_l \in \mathbb{R}^{D_l}$ in a layer $l$ should "be predictive of" those in either the previous layer $\mathbf{h}_{l-1} \in \mathbb{R}^{D_{l-1}}$ or subsequent layer $\mathbf{h}_{l+1} \in \mathbb{R}^{D_{l+1}}$, via a parametric non-linear transformation $\mathbf{f}(\cdot, \theta_l)$. Applying this principle in e.g. the feedforward direction, together with the Gaussian assumption, suggests the following form for the factor energy:

$$E(\mathbf{h}_l, \mathbf{h}_{l-1}, \theta_l) = \frac{\|\mathbf{h}_l - \mathbf{f}(\mathbf{h}_{l-1}, \theta_l)\|_2^2}{2\sigma_l^2} , \tag{4.1}$$

which is low when $\mathbf{f}(\mathbf{h}_{l-1}, \theta_l)$ matches the output[1] $\mathbf{h}_l$ using parameters $\theta_l$ and input $\mathbf{h}_{l-1}$. $\sigma_l$ is the factor strength ($\Lambda_{\mathbf{y}_l} = \frac{1}{\sigma_l^2} I_{D_{l-1}}$). Through choice of $\mathbf{f}(\cdot, \cdot)$ we can encode different operations and inductive biases.

For example, we could choose $\mathbf{f}(\cdot, \theta_l)$ to be a simple dense projection, giving an inter-layer

[1] We use "inputs" to mean the activation variables within a layer which are closer to the pixels (with smaller $l$), and "outputs" those which are further away. However, BP is bidirectional so these quantities are not equivalent to the inputs of a function.

factor with energy

$$E_{\text{dense}}\left(\mathbf{h}_{l-1}, \mathbf{h}_l, \theta_l\right) = \frac{\left\|\mathbf{h}_l - g\left(W_l^\intercal \mathbf{h}_{l-1} + \mathbf{d}_l\right)\right\|_2^2}{2\sigma_l^2} \ , \tag{4.2}$$

where the layer parameters $\theta_l = (W_l, \mathbf{d}_l)$ and $g(\cdot)$ is an elementwise non-linear activation function.

The dense projection layer energy (4.2) depends on a large number — $\mathcal{O}\left(\dim \mathbf{h}_{l-1} \cdot \dim \mathbf{h}_l\right)$ — of variables, which implies a highly connected factor graph (we later show it can be decomposed, see Section 4.2.3.2). In contrast, convolutional neural networks (CNNs), which have been hugely successfully in computer vision, have a sparse connectivity structure, suggesting potential factor graph analogues which admit efficient inference (as each factor connects to a small number of variables). In our convolutional layers, a factor at spatial location $(a, b)$ connects to i) the $K_l \times K_l$ patch of the input within its receptive field, $H_{l-1}^{(a,b)} \in \mathbb{R}^{K_l \times K_l \times C_{l-1}}$, ii) the corresponding activation variable for an output channel $c$, $h_l^{(a,b,c)} \in \mathbb{R}^{C_l}$ and iii) the parameters $\theta_l$: filters $W_l^{(c)} \in \mathbb{R}^{K_l \times K_l \times C_{l-1}}$ and bias $b_l^{(c)}$ for output channel $c$, which are shared across the layer. The factor energy is

$$E_{\text{conv}}^{(a,b,c)} = \frac{\left(h_l^{(a,b,c)} - r\left(H_{l-1}^{(a,b)}, W_l^{(c)}, b_l^{(c)}\right)\right)^2}{2\sigma_l^2} \ , \tag{4.3}$$

$$r\left(\mathbf{A}, \mathbf{B}, s\right) := g\left(\text{vec}\left(A\right) \cdot \text{vec}\left(B\right) + s\right) \ . \tag{4.4}$$

We have removed the functional dependence of $E_{\text{conv}}^{(a,b,c)}$ on the connected variables for brevity. The total energy of the layer is found by summing the values of $E_{\text{conv}}^{(a,b,c)}$ over the spatial locations $(a, b)$ and output channels $c$.

We can similarly define a transposed convolution layer. In this case, each filter is weighted by the output variable of its corresponding channel, and the weighted sum reconstructs the inputs. Note that each input variable $h_{l-1}^{(a,b,c)}$ will be inside the receptive fields of many output variables $H_l^{(a,b)}$ whose contributions are summed to give its reconstruction. For example, for a stride of one and neglecting edge effects, these contributions can be combined according to:

$$E_{\text{convT}}^{(a,b,c)} = \frac{\left(h_{l-1}^{(a,b,c)} - r\left(H_l^{(a,b)}, W_l^{(c)}, b_l^{(c)}\right)\right)^2}{2\sigma_l^2} \ . \tag{4.5}$$

$r(\cdot)$ is defined above (4.4). In this case, the parameters have dimension $W_l^{(c)} \in \mathbb{R}^{K_l \times K_l \times C_l}$ and $b_l^{(c)} \in \mathbb{R}$.

Note that the transposed convolution factor $E_{\text{convT}}$ is *implicitly* generative: the parameters $\theta_l$ combine with the deeper layer activations $H_l$ to reconstruct the activations closer to the pixels $H_{l-1}$. In contrast, the "feedforward" dense (4.2) and convolutional (4.3) layers are implicitly discriminative. However, the directionality of the energy function is only a "soft" specification as the factor graph and GBP inference are undirected.

In addition to convolutional and dense factors, we design factors akin to other common CNN layers. For example, we use max-pooling factors to reduce the spatial extent of the representation,

$$E_{\text{maxpool}}^{(a,b,c)} = \frac{1}{2\sigma_l^2}\left(\max\left(H_{l-1}^{(a,b,c)}\right) - h_l^{(a,b,c)}\right)^2 , \tag{4.6}$$

upsampling layers to increase it

$$E_{\text{upsample}}^{(a,b,c)} = \frac{1}{2\sigma_l^2}\sum_{i,j=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor}\left(h_{l-1}^{(a-i,b-j,c)} - h_l^{(a,b,c)}\right)^2 , \tag{4.7}$$

and softmax observation factors for class supervision

$$E_{\text{softmax}} = \frac{\|\text{softmax}(\mathbf{h}_L) - \mathbb{1}_y\|_2^2}{2\sigma_l^2} , \tag{4.8}$$

where $\mathbb{1}_y$ is the one-hot vector, with a value 1 at the $y$th element. Note the the set of layers described here is non-exhaustive, we leave the exploration of other layer types as future work.

These "layer" abstractions can be composed to produce deep models with similar design freedom to DL. Further, our models may be overparameterised by introducing large numbers of learnable weights and we believe that the inclusion of non-linear activation functions $g(\cdot)$ in our factor graph can aid representation learning as in DL. In particular, we note that for non-linear factors, the factor Jacobian is a function of the current variable estimates $J_j = J_j(\mathbf{x}_{j,0})$, which will cause the strength of the factors (2.77) to vary depending on the input data. We expect this to produce a similar "soft switching" of connections as observed in non-linear NNs.

### 4.2.2 Learning and Prediction with GBP Inference

We have described the components of deep factor graph models, in which inputs, outputs, activations and parameters are random variables. As these models include non-linear factors such as (4.2), (4.3) and (4.5), we use the iterative linearisation scheme described in Section 2.4.4.2 to do approximate inference with GBP. We apply this inference engine to estimate posterior marginals for all latent variables. Note that there is no fundamental difference between training and prediction — only a difference in which variables are observed. In training, we infer a posterior over parameters given observed inputs and, where supervision is available, outputs. To then make predictions on new examples, we run GBP to predict the unobserved inputs/outputs given the observed inputs/outputs and parameters.

Our message schedule proceeds as follows unless stated otherwise. For a given batch, we initialise the graph and update messages by sweeping forward and backward through the layers: first nearest the input observations, progressing to the deepest layer and back again. We repeat this for a specified number of iterations. Within each layer, we compute all factor-to-variable updates in parallel, and likewise for the variable-to-factor messages of each variable type (inputs, outputs, parameters as applicable). In addition, we experimentally show our approach works well with layerwise random message schedules (Section 4.3.3.3). While we have demonstrated these schedules work, they are likely suboptimal and we leave exploration for future work. We find that applying damping [Murphy et al., 2013] and dropout to the factor-to-variable messages is sufficient for stable GBP.

### 4.2.3 Efficient GBP

#### 4.2.3.1 Optimised Linear Algebra

Efficient inference is necessary for our models to scale to interesting problems. However, the inversion of $\Lambda^{(\backslash i,\backslash i)}_{k+m_{\backslash i}} \in \mathbb{R}^{(|\text{nei}(\phi_k)|-1)\times(|\text{nei}(\phi_k)|-1)}$ in the factor-to-variable update, (2.66) and (2.67), has $\mathcal{O}\left(|\text{nei}(\phi_k)|^3\right)$ complexity for each outgoing message, bottlenecking GBP. Below, we briefly outline our algebraic optimisations for faster factor-to-variable updates. We provide a more detailed exposition in Appendix B.1.

**Optimised inversion** We note that observation dimensions $M_k = \dim \mathbf{y}_k$ are typically smaller than the number of variables connected to a factor $M_k < |\text{nei}\,(\phi_k)\,|$. In this case we can alleviate the inversion bottleneck by exploiting the structure of the matrix being inverted.

In particular, the matrix to invert is

$$\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)} = \Lambda_k^{(\setminus i,\setminus i)} + \Lambda_{m_{\setminus i}}, \tag{4.9}$$

where $\Lambda_k^{(\setminus i,\setminus i)}$ is the precision matrix with the row and column for $x_i$ removed, and $\Lambda_{m_{\setminus i}}$ is the diagonal matrix of incoming messages from all variables except recipient $x_i$. The full precision for a linear or linearised factor can be written as

$$\Lambda_k = J_k^\intercal \Lambda_{\mathbf{y}_k} J_k \tag{4.10}$$

and therefore the precision with the row and column for $x_i$ removed is

$$\Lambda_k^{(\setminus i,\setminus i)} = \left(J_k^{(:,\setminus i)}\right)^\intercal \Lambda_{\mathbf{y}_k} J_k^{(:,\setminus i)}. \tag{4.11}$$

This is low-rank if $M_k = \dim \mathbf{y}_k < |\text{nei}\,(\phi_k)\,|$. Thus we see that the target for inversion (4.9) is the sum of a diagonal matrix of messages $\Lambda_{m_{\setminus i}}$ and a low-rank partial precision $\Lambda_k^{(\setminus i,\setminus i)}$. We can exploit this using the Woodbury identity [Woodbury, 1950], which gives

$$\left(\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)}\right)^{-1} = \Lambda_{m_{\setminus i}}^{-1} - \Lambda_{m_{\setminus i}}^{-1}(J_k^{(:,\setminus i)})^\intercal \left(\Lambda_{\mathbf{y}_k}^{-1} + J_k^{(:,\setminus i)}\Lambda_{m_{\setminus i}}^{-1}(J_k^{(:,\setminus i)})^\intercal\right)^{-1} J_k^{(:,\setminus i)}\Lambda_{m_{\setminus i}}^{-1}. \tag{4.12}$$

In this form, we only need to invert diagonal matrices of size $|\text{nei}\,(\phi_k)\,|$ and dense matrices of size $M_k$. The resulting complexity is $\mathcal{O}\left(|\text{nei}\,(\phi_k)\,|M_k^2 + M_k^3\right)$.

**Better intermediates** We can improve further by computing intermediates for all outgoing messages upfront, rather than computing each message from scratch. Specifically, if we substitute the optimised inversion (4.12) back into the factor-to-variable updates (2.66) and (2.67), and rearrange the result (see Appendix B.1), we can arrive at

$$\eta_{\phi_k \to x_i} = \eta_k^{(i)} - \left(J_k^{(:,i)}\right)^\intercal \Lambda_{\mathbf{y}_k}\left(T_i - U_i\left(\Lambda_{\mathbf{y}_k}^{-1} - U_i\right)^{-1} T_i\right) \tag{4.13}$$

$$\Lambda_{\phi_k \to x_i} = \Lambda_k^{(i,i)} - \left(J_k^{(:,i)}\right)^\intercal \Lambda_{\mathbf{y}_k}\left(U_i - U_i\left(\Lambda_{\mathbf{y}_k}^{-1} - U_i\right)^{-1} U_i\right)\Lambda_{\mathbf{y}_k} J_k^{(:,i)}\ , \tag{4.14}$$

where we have defined $U_i := J_k^{(:,\setminus i)}\Lambda_{m_{\setminus i}}^{-1}\left(J_k^{(:,\setminus i)}\right)^\intercal$ and $T_i := J_k^{(:,\setminus i)}\Lambda_{m_{\setminus i}}^{-1}\eta_{k+m_{\setminus i}}^{(\setminus i)}$.

Computation of each $U_i, T_i$ term directly is $\mathcal{O}\left(|\text{nei}(\phi_k)|M_k^2\right)$ complexity ($\Lambda_m$ is diagonal). However, if we first compute $U := J_k\Lambda_m^{-1}(J_k)^\intercal$ and $T := J_k\Lambda_m^{-1}\eta_{k+m}$ for all outgoing messages, then we can cheaply calculate the $U_i = U - J_k^{(:,i)}\Lambda_{m_i}^{-1}(J_k^{(:,i)})^\intercal$ and $T_i = T -$

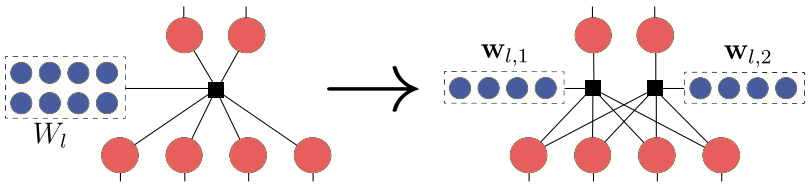



Figure 4.2: **An example of inter-layer factor decomposition**, for a dense layer with 4 inputs and 2 outputs. Decomposing one highly connected factor into multiple, sparsely connected ones can enable more efficient GBP. Biases not shown for clarity.

$J_k^{(:,i)}\Lambda_{m_i}^{-1}\eta_{k+m_{\setminus i}}^{(i)}$ for each outgoing message. As a result we can compute *all* outgoing messages from a factor with complexity $\mathcal{O}\left(|\text{nei}(\phi_k)|M_k^3\right)$.

In summary, these optimisations change the time complexity of updating messages from a factor to all $|\text{nei}(\phi_k)|$ variables, from $\mathcal{O}(|\text{nei}(\phi_k)| \cdot |\text{nei}(\phi_k)|^3)$ to $\mathcal{O}\left(|\text{nei}(\phi_k)|M_k^3\right)$. Space complexity is changed from $\mathcal{O}(|\text{nei}(\phi_k)|^2)$ to $\mathcal{O}(|\text{nei}(\phi_k)|M_k + M_k^2)$. Again, we refer the reader to Appendix B.1 for details.

#### 4.2.3.2 Factor Decomposition

The above results constitute significant savings when $M_k << |\text{nei}(\phi_k)|$, raising the question of typical values of dimension $M_k$. For feedforward factors such as (4.3) and (4.2), $M_k$ is equal to the number of output variables connected to the factor. Thus for layers with many outputs, even the optimised factor-to-variable updates may be prohibitive, due to the cubic complexity. However, we note that such factors can be decomposed along the output dimension as illustrated in Figure 4.2. For example, a dense factor with energy (4.2) may

be decomposed into $M_k$ smaller factors, one per output variable. The resulting energy is

$$E_{\text{dense}}\left(\mathbf{h}_{l-1}, \mathbf{h}_l, \theta_l\right) = \frac{\left\|\mathbf{h}_l - g\left(W_l^\intercal \mathbf{h}_{l-1} + \mathbf{d}_l\right)\right\|_2^2}{2\sigma_l^2} \tag{4.15}$$

$$= \sum_{j=1}^{\dim(\mathbf{h}_l)} \frac{\left\|h_{l,j} - g\left(\mathbf{w}_{l,j}^\intercal \mathbf{h}_{l-1} + d_{l,j}\right)\right\|_2^2}{2\sigma_l^2} \tag{4.16}$$

$$= \sum_{j=1}^{\dim(\mathbf{h}_l)} E_{\text{dense},j}\left(\mathbf{h}_{l-1}, h_{l,j}, \theta_{l,j}\right) \tag{4.17}$$

$$E_{\text{dense},j}\left(\mathbf{h}_{l-1}, h_{l,j}, \theta_{l,j}\right) := \frac{\left\|h_{l,j} - g\left(\mathbf{w}_{l,j}^\intercal \mathbf{h}_{l-1} + d_{l,j}\right)\right\|_2^2}{2\sigma_l^2}\ , \tag{4.18}$$

where $\mathbf{w}_{l,j}$ is the $j$th column of $W_l$ and $d_{l,j}$ is the $j$th element of $\mathbf{d}_l$. As the energy of the original dense factor is recovered by summing the decomposed factor energies, the model is remains unchanged. However, each message update is $M_l^2$ more efficient because one factor with $M_l$ outputs has been replaced by $M_l$ factors each with one output.

We now consider how the factor-to-variable message update complexity translates to the complexity of updating *all* the messages in a model. As an illustrative example, we consider a model comprising $L$ dense layers, each with $C$ input units and $C$ output units. As described, we can decompose the factor between each pair of layers into $C$ smaller factors, each with $M = 1$ and connected to $2 \cdot (C + 1)$ variables. The complexity of factor-to-variable message updates in a layer is therefore $\mathcal{O}(C^2)$, the same as the variable-to-factor message updates. In the non-linear case, we must also account for the computation of the factor Jacobian $J$ and information vector $\eta$ each time the factor is relinearised. This requires the matrix-vector product $W_l^\intercal \mathbf{h}_{l-1}$, which is also complexity $\mathcal{O}(C^2)$. We thus conclude that the overall complexity for updating all messages in a model of $L$ such layers, with a batch size of $B$, is $\mathcal{O}(BLC^2)$. This is the same complexity as a forward or backwards pass of backpropagation in the equivalent MLP, and the same as that for the message passing approach of Lucibello et al. [2022].

### 4.2.4 Continual Learning and Minibatching

We now describe our approach to continual learning of model parameters with Bayesian filtering. For generality, we use $\theta = \{\theta_l\}_{l=1}^L$ to denote the set of all parameters where $\theta_l$ is the

vector of those for layer $l$. After initialising parameter priors$\{$ $\{p_{t=1}(\theta_{l,i}) \leftarrow \mathcal{N}(0, \sigma_l); i = 1, \ldots, \dim \theta_l; l = 1, \ldots, L\}$ we perform the following for each task $t$ in a sequence of datasets $[\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T]$:

1. construct a copy of the graph with task dataset $\mathcal{D}_t$,
2. update the unary prior on each parameter variable, setting it to the marginal posterior estimate from the previous task $p_t(\theta_{l,i}) \leftarrow p(\theta_{l,i}|\mathcal{D}_{1:t-1})$,
3. run GBP training to get an estimate of the updated posterior $p(\theta|\mathcal{D}_{1:t}) = \prod_l \prod_{i=1}^{\dim \theta_l} p(\theta_{l,i}|\mathcal{D}_{1:t})$.

This method is equivalent to doing message passing in the combined graphical model for all tasks, but where messages between tasks are only passed forward in $t$. The advantage however, is that datapoints can be discarded after processing, and the combined graphical model for all tasks does not have to be stored in memory. As such, we also use this routine for memory-efficient training by dividing the dataset into minibatches and treating each minibatch as a task.

In the case of exact inference, the resulting posterior from the incremental learning method above is equal to the batch posterior (see discussion in Section 1.2.2.1, point 3). However, our networks contain cycles and non-linear activations, and as a result our inference is only an approximation. In particular, GBP marginal estimates are known to be overconfident in loopy graphs [Weiss and Freeman, 1999], which could potentially lead to a lack of plasticity when fitting future data. Nonetheless, we find this simple routine works well in practice.

## 4.3 Experiments

We demonstrate our approach with four sets of experiments: small regression tasks (Section 4.3.1), sequential video denoising (Section 4.3.2.1), image classification (Section 4.3.3) and a predictive control example (Section 4.3.4). Our TensorFlow [Abadi et al., 2015] implementation is made available[2].

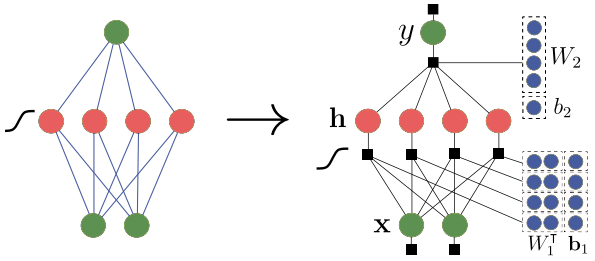
| | **XOR** | | | **Regression** | | |
|---|---|---|---|---|---|---|
| Layer # | 1 | 2 | 3 | 1 | 2 | 3 |
| Type | Dense | Dense | Softmax | Dense | Dense | Output obs. |
| Input dim. | 2 | 8 | 2 | 1 | 16 | 1 |
| Output dim. | 8 | 2 | 2 | 16 | 1 | 1 |
| Inc. bias | ✓ | ✓ | - | ✓ | ✓ | - |
| $\theta$ prior $\sigma$ | 3.0 | 3.0 | - | 6.0 | 1.5 | - |
| $\mathbf{h}$ prior $\sigma$ | 5.0 | 2.0 | - | 10.0 | 3.0 | - |
| $\mathbf{x}$ obs. $\sigma$ | 0.02 | - | - | 0.02 | - | - |
| $y$ obs. $\sigma$ | - | - | 0.1 | - | - | 0.05 |
| Inter-layer $\sigma$ | 0.1 | 0.1 | - | $5 \times 10^{-3}$ | $1 \times 10^{-2}$ | - |
| Activation | Leaky ReLU | Linear | - | Sigmoid | Linear | - |

Table 4.1: **MLP-like factor graph model specifications** used for the XOR and regression experiments. For XOR we run for 600 iterations with a damping factor 0.7, and for regression we use $4 \times 10^3$ iterations and 0.8 damping.

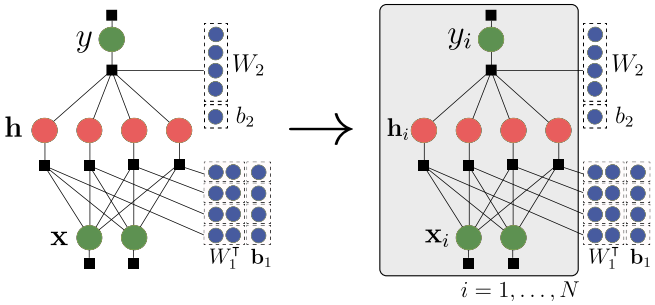
### 4.3.1 Toy Experiments

To first understand how to apply our method in practice, we will use it to address two toy learning problems. The first is "Exclusive-OR" (Figure 4.4a), which is a well-known minimal test for non-linear modelling [Minsky and Papert, 1969], and comprises a dataset of only 4 observations. The second is a one-dimensional non-linear regression problem (Figure 4.4b) with 90 observations.

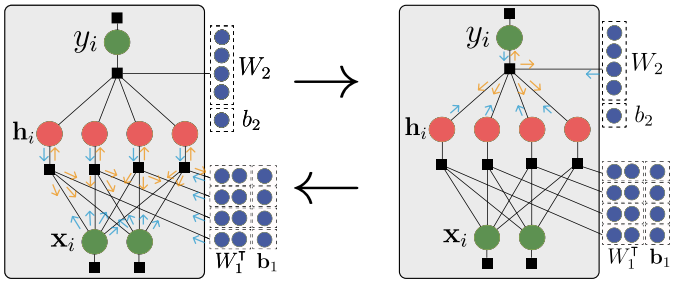
[2] github.com/sethnabarro/gbp_learning/



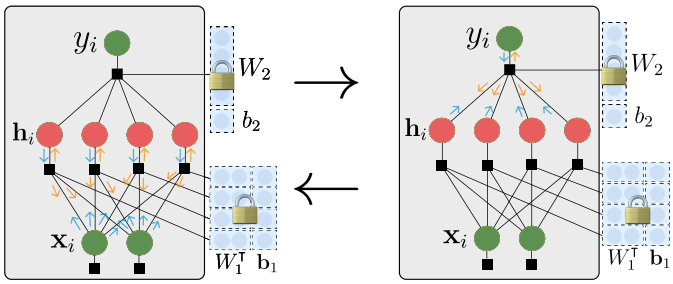

(a) Model design




(b) Model construction




(c) Training with layerwise message updates




(d) Predictions with fixed weights, no output observation factor

Figure 4.3: **Method for applying GBP Learning to a toy problem.** We begin with the design of the model, which may a translation from a neural network to a factor graph as in (a). $\int$ denotes non-linear activation function. The model is instantiated, with one copy of input, output and activation variables per data example, while weight variables are shared (b). We then carry out training and prediction using GBP with layerwise message schedules (c, d). Orange arrows represent factor-to-variable messages, blue arrows variable-to-factor.

In both cases, we break GBP Learning down into four steps:

1. **Model design,** Figure 4.3a. Usually the model is a factor graph analogue of a neural network, in this case we use single hidden layer MLPs for both tasks. The full model details are presented in Table 4.1.

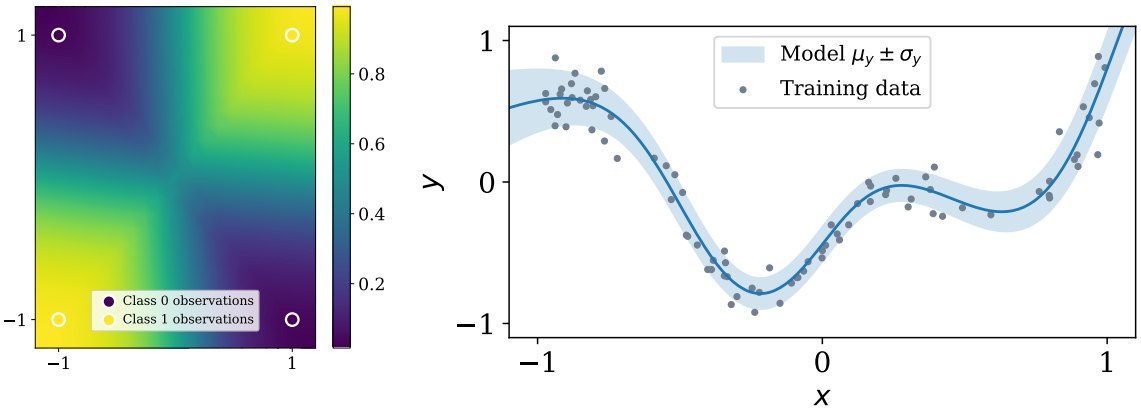
   In addition to the standard NN hyperparameters such as architecture and activation function, we must also choose the strengths $\sigma$ of the observation, prior and inter-layer factors. We generally choose these via a crude grid-search, but usually find that good values include high strength inter-layer factors, and observation factor strengths which roughly match aleatoric uncertainty.

2. **Model construction,** Figure 4.3b. Here we initialise the model by creating instantiating the necessary factors and edges. The parameters are shared over all observations, where each of the input, output and activation variables are replicated, once per data example.

3. **GBP training,** Figure 4.3c. After the model has been initialised, we run GBP to infer the parameter posterior. We use a layerwise message schedule, and damping of the factor-to-variable messages to stabilise GBP as described in Section 4.2.2.

4. **GBP prediction**, Figure 4.3d. After training we fix the values of the parameters, run GBP to infer the activations and output variables for given input observations. We note that, in theory, we could instead attach posterior factors to approximately constrain the parameters during prediction. However, in practice we find fixing their values produces more accurate predictions.

It is clear from the results in Figure 4.4 that we are able to fit (simple) nonlinear functions, indicating that the linearisation approach described in Section 2.4.4.2 is effective. We further note that the different activation functions used result in different functional behaviours: the leaky ReLU used for XOR results in a piecewise prediction for each of the four quadrants, whereas the sigmoid used for one-dimensional regression gives a smoothly varying prediction. While this is not-surprising, it is reassuring that our method is able to recover similar inductive biases seen in NNs.




(a) XOR, $p(y^* = 1|\mathbf{x}^*, \theta, \mathcal{D})$ (b) Nonlinear regression

Figure 4.4: **Fitted models for toy experiments.** GBP Learning in MLP-like factor graphs (Figure 4.3a) can solve nonlinear regression and classification tasks.

### 4.3.2 Video Denoising

We now ask whether the learnable components in our model can improve performance over a hand-designed solver. We apply our method to the task of denoising the "bear" video from the DAVIS dataset [Perazzi et al., 2016], downsampled to $258 \times 454$ with bilinear interpolation. We sample 10% of the pixels in each frame of the video, replacing their intensities with noise drawn from $U(0, 1)$. Performance is assessed by how well the denoised image matches the ground truth under the peak signal-to-noise ratio (PSNR).

The estimated pixel intensities may explained by the noisy pixel observations and/or the model reconstruction. Both the pixel observation and reconstruction factors have robust energies (see Section 2.4.4.2) to enable the pixel variables to "switch" between these explanations.

**Baseline setup** We evaluate two types of reconstruction algorithm. As a baseline we run a hand-designed smoothing model which we call *pairwise smoothing*. Like ours, it is a factor graph model over pixels, in which GBP is used to is used to infer denoised intensities. Each pair of neighbouring pixels are connected via smoothing factors which encourage their intensities to be similar (see e.g Ortiz et al. [2021]). Robust energies on the smoothness factors aid the preservation of edges present in the original image.

This classical baseline represents a reasonable "factor graph + GBP" denoiser, with

similar computational properties to ours but without the ability to learn from data. The comparison thus quantifies the benefit of our proposal to incorporate learning into factor graph inference.

**GBP Learning setup** The second set of reconstruction algorithms we evaluate is GBP Learning in convolutional factor graphs. We examine the impact of model depth by comparing: i) a single transposed convolution layer (factor energies as per (4.5)) with four filters, and ii) a five layer model comprising transposed convolution and upsampling layers.

For both the single and five layer models, we test two routines for parameter learning. First, we learn the filters from scratch on each frame. We refer to this as *per-frame* learning. Second, we use the continual learning routine described in Section 4.2.4, carrying over the parameter posterior from the previous frame as a prior for the next, iterated over the course of the video.

We highlight that with GBP learning we seek to infer *both* the true pixel intensities *and* the model parameters, from the *noisy images only* and without supervision. The reconstruction factors in noiseless regions incentivise the learning of parameters which are consistent with the true pixel statistics. These learnt parameters provide a means to remove the noise elsewhere.

The hyperparameters for all models were tuned using the first five frames of the video, and the details of the final models are given in Tables 4.2 and 4.3. Denoising the entire 82-frame video with the single learnable layer model took $\sim$ 8mins on a NVIDIA RTX 3090 GPU, and the five layer model took $\sim$ 27mins. Pairwise smoothing took $\sim$ 2mins.

#### 4.3.2.1 Results

The PSNR results are presented in Figure 4.5, along with examples of denoised frames in Figure 4.7 and enlarged crops in Figure 4.8. All methods exhibit a similar relative evolution of PSNR over the course of the video, slowly rising until the fiftieth frame and then falling again. We conjecture that this is due to varying image content, with some frames being easier to denoise than others.

The models with learnable parameters (orange and green) significantly outperform the classical baseline (blue). We attribute this to the learned models being able to capture image structure, which enables more high-frequency features to be retained while removing the corruption. Indeed, we can see that even with robust factors with tuned robust thresholds,

| | **1 Layer** | **5 Layer** | | | | |
|---|---|---|---|---|---|---|
| Layer # | 1 | 1 | 2 | 3 | 4 | 5 |
| Layer type | ConvT | ConvT | Upsample | ConvT | Upsample | ConvT |
| Num. filters | 4 | 4 | - | 8 | - | 8 |
| Inc. bias | ✓ | ✓ | - | ✓ | - | ✓ |
| Kernel size | $3 \times 3$ | $3 \times 3$ | $2 \times 2$ | $3 \times 3$ | $2 \times 2$ | $3 \times 3$ |
| Conv. recon. $\sigma$ | 0.1 | 0.12 | 0.03 | 0.07 | 0.03 | 0.07 |
| Recon $E_{\text{rob}}$ | 2.0 | 2.5 | - | - | - | - |
| $\theta$ prior $\sigma$ | 0.018 | 0.15 | - | 0.3 | - | 0.3 |
| $\mathbf{h}$ prior $\sigma$ | 0.5 | 0.5 | - | 0.5 | - | 0.5 |
| Pixel obs. $\sigma$ | 0.2 | 0.2 | - | - | - | - |
| Pixel obs $E_{\text{rob}}$ | 0.04 | 0.2 | - | - | - | - |
| Activation $g(\cdot)$ | Linear | Linear | - | Leaky ReLU | - | Leaky ReLU |

Table 4.2: **The convolutional factor graph models used for the video denoising experiments**. "ConvT" denotes transposed convolution layer. $E_{\text{rob}}$ is the "robust threshold": the value of the factor energy above which it is in the robust region. For the Tukey robust factors [Tukey, 1960] we use, the energy is constant after the robust threshold. For the single layer model, we run GBP for 300 iterations per frame, factor-to-variable damping of 0.8, and dropout of 0.6. In the five layer model we use 500 iterations and the same damping and dropout.

the pairwise smoother is unable to effectively remove much of the corruption (Figure 4.8g) in comparison to e.g. the single layer learnt model (Figure 4.8c). For higher robust thresholds, the pairwise smoother results in blurred reconstructions which destroy the high-frequency features of the image.

We see a clear benefit of depth, with significantly higher PSNR scores for the five layer model (orange/green dotted) over the single layer (orange/green dash). This can be seen qualititatively, by visually comparing the reconstructions (e.g. Figure 4.8d and Figure 4.8f) or residuals (e.g. Figure 4.9b and Figure 4.9d).

Despite the single layer model having only four filters, we find that it sometimes learns to reconstruct noise when learning continually over the video. To counteract this, we set the prior on filters for each frame to be a crude interpolation between the previous posterior

| Layer # | 1 |
|---|---|
| Layer type | Pairwise smoothing |
| Pixel obs. $\sigma$ | 0.2 |
| Pixel obs $E_{\text{rob}}$ | 0.02 |
| Pairwise $\sigma$ | 1.3 |
| Pairwise $E_{\text{rob}}$ | 0.12 |

Table 4.3: **The pairwise smoothing baseline model used for the video denoising experiments.** $E_{\text{rob}}$ is the "robust threshold" – see caption Table 4.2 caption for details. We use 200 GBP iterations per frame and factor-to-variable message damping of 0.7.

and the original prior. Specifically,

$$p_0(\theta_i) = \mathcal{N}(\mu_{i,0}, \sigma_{i,0}) \quad \text{(original prior)} \tag{4.19}$$

$$p(\theta_i | X_1, \dots X_{t-1}) = \mathcal{N}(\mu_{i,t-1}, \sigma_{i,t-1}) \quad \text{(previous posterior)} \tag{4.20}$$

$$p'(\theta_i | X_1, \dots X_{t-1}) = \mathcal{N}\left(\mu'_{i,t-1}, \sigma'_{i,t-1}\right) \quad \text{(prior for next frame)} \tag{4.21}$$

$$\mu'_{i,t-1} = \alpha \cdot \mu_{i,t-1} + (1-\alpha) \cdot \mu_{i,0} \tag{4.22}$$

$$\sigma'_{l,i} = \alpha \cdot \sigma_{i,t-1} + (1-\alpha) \cdot \sigma_{i,0} \ . \tag{4.23}$$

We used interpolation weight $\alpha = 0.5$. The results presented here for the incrementally learnt, single-layer model (in e.g. Figures 4.5 and 4.7) are using this interpolation.

In contrast, the deeper model does not require additional regularisation when trained continually. These results imply i) lower layers in deep factor graphs are effectively regularised by higher level feature learning; ii) GBP Learning is can find aligned multi-layer representations with only local message updates.

In general we see that continual learning (green) leads to an improvement over learning per-frame (orange), suggesting that we can effectively fuse what has been learnt in previous frames and use it to better denoise the current frame. The difference is clear in the single layer model in which some features of the image are not properly reconstructed with per-frame learning. For example, a patch of fur on the bear's back is somewhat discoloured in the per-frame case (top, centre-right of crop in Figure 4.8c and corresponding residual in Figure 4.9a), which is not true of the continually learnt single layer model. Note the single layer model with continual learning uses the interpolation scheme (4.22) and (4.23). The benefit of incremental learning is less pronounced in the five layer model, but we can see more corrupted pixels remaining in the per-frame case (Figure 4.9c vs Figure 4.9d).

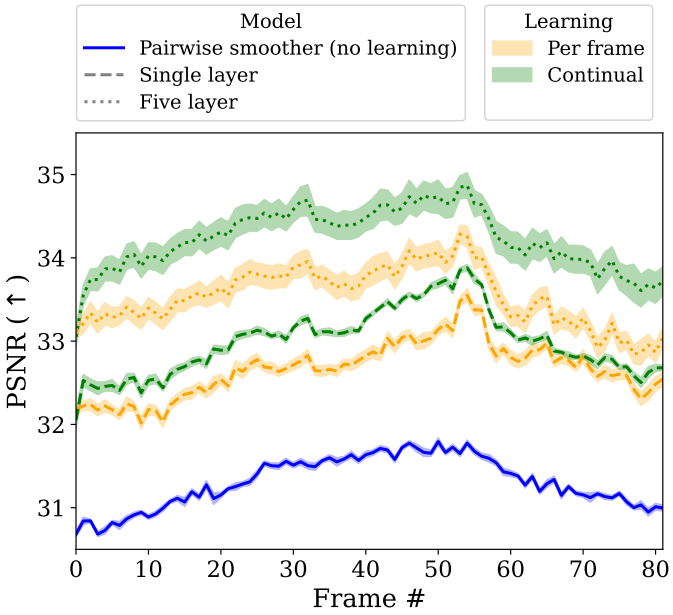



Figure 4.5: **Evolution of PSNR scores during video denoising.** Factor graphs with learnable components outperform a hand-specified pairwise smoother. Continual learning of parameters over the video further improves the PSNR over per-frame learning, and the deep model outperforms the single layer. Shading is ±1 standard error (SE) over 10 seeds.

The histogram of residuals over all image pixels (Figure 4.6) is in agreement with the quantitative and qualitative results discussed. In particular, the tails of the residual distributions show that i) the pairwise smoother has most mass far from zero, ii) the per-frame models have heavier residual tails than their continual counterparts (most pronounced for negative residuals), iii) single layer model residuals have heavier tails than five layer models.

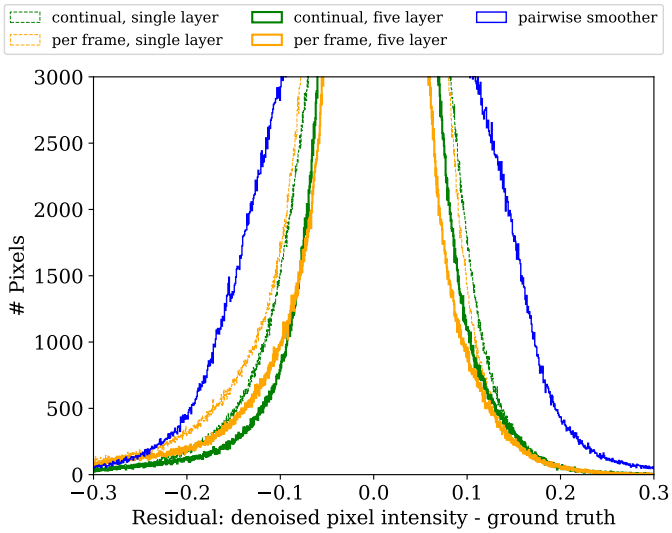



Figure 4.6: **Histogram of residuals for all pixels in frame 5.** The tails of the histogram of errors shows i) the pairwise smoother residuals have the greatest variance as expected, ii) the tails of the residuals for single layer models are heavier than for five layer models, iii) the tails of the residuals for the continually learnt models are heavier than those for per-frame learning (particularly in the negative region).

(a) Clean image

(b) Corrupted image

(c) Per-frame learning, single layer, PSNR = 32.6

(d) Continual learning, single layer, PSNR = 32.9

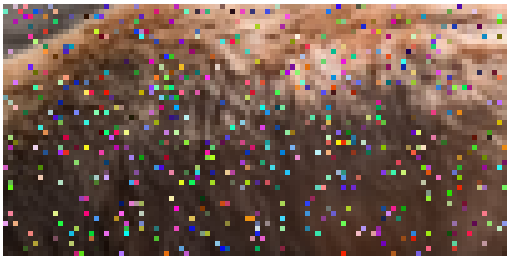
(e) Per-frame learning, five layer, PSNR = 33.4

(f) Continual learning, five layer, PSNR = 34.3

(g) Pairwise smoothing, PSNR = 31.1

Figure 4.7: **Video frame 5 denoised by each method.** The red rectangle in (a) covers the cropped region presented in Figure 4.8. Best viewed digitally for zoom.

(a) Clean image



(b) Corrupted image

(c) Per-frame learning, single layer

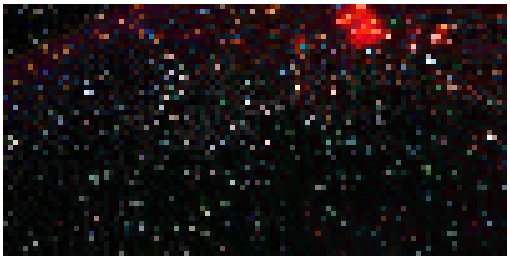
(d) Continual learning, single layer

(e) Per-frame learning, five layer

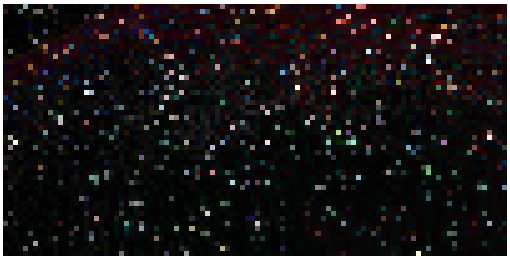
(f) Continual learning, five layer

(g) Pairwise smoothing

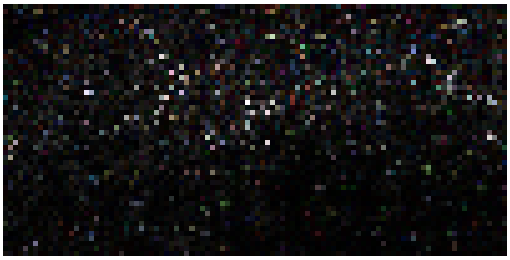
Figure 4.8: **A crop from video frame 5.** The learnt models are able to remove more noise while retaining more high-frequency signal. The location of the crop in the full image is marked with a red rectangle in Figure 4.7a.



(a) Per-frame learning, single layer



(b) Continual learning, single layer



(c) Per-frame learning, five layer

(d) Continual learning, five layer

Figure 4.9: **Absolute residuals from the crop from frame 5.** The five layer models remove more of the corruption than the single layer ones. Further, continually learnt models outperform per-frame learning. The location of this crop in the full image is marked with a red rectangle in Figure 4.7a.

### 4.3.3 Image Classification

Next, we assess our method in a supervised learning context, evaluating it on the MNIST dataset[3]. The goal here is to gauge both sample efficiency and the ability to learn continually with only one pass through the training set. To evaluate the former, we vary the size of the training set.

**GBP Learning setup** We train a convolutional factor graph with GBP Learning. The architecture is similar to Figure 4.1: a convolutional layer (with energy as per (4.3)), followed by a max-pool (4.6), dense layer (4.2) and (at train time) a class observation factor (4.8) on the output logit variables. Training is minibatched via the Bayesian filtering approach of Section 4.2.4, so the model only sees each datapoint once, after which it can be discarded. On a held-out validation set we tune the factor strengths, the dimensions of each layer, and the amount of damping and dropout to apply to factor-to-variable message updates.

[3] yann.lecun.com/exdb/mnist/

| Layer # | 1 | 2 | 3 | 4† |
|---|---|---|---|---|
| Layer type | Conv. | Max pool | Dense | Softmax |
| Num. filters | 16 | - | - | - |
| Kernel size | $5 \times 5$ | $2 \times 2$ | - | - |
| Dense num. inputs | - | - | 2304 | - |
| Dense num. outputs | - | - | 10 | - |
| Inc. bias | ✓ | - | ✓ | - |
| $\theta$ prior $\sigma$ | 0.1 | - | 0.15 | - |
| $\mathbf{h}$ prior $\sigma$ | 3.0 | 3.0 | 2.0 | - |
| Recon. $\sigma$ | 0.01 | 0.01 | 0.01 | - |
| Activation $g(\cdot)$ | Leaky ReLU | Linear | Linear | - |
| Class observation $\sigma$ | - | - | - | 0.01 |

Table 4.4: **The convolutional factor graph model used for the MNIST experiment.** The dimensions of the layers are chosen such that no padding is necessary. Note that "Recon $\sigma$" denotes the strength of the factors which connect one layer to the next. †Softmax layer only included at training time, when class observation is available. At test time, the final state of the last layer variables after GBP inference is treated as the logit prediction. We run 500 GBP iterations on each training batch of 50 examples, with factor-to-variable damping of 0.9 and dropout of 0.5.

**Baseline setup** We compare against two baselines:

1. **A linear classifier baseline** which predicts logits via a dense projection of the image vector, trained with Adam [Kingma and Ba, 2014] over multiple epochs. We tune the learning rate and the number of epochs on a validation set.

2. **The CNN equivalent** of our factor graph, trained with Adam for a single epoch, with with a FIFO replay buffer (of varying sizes). For each buffer size we tune the following hyperparameters:

    - Number of elements in each batch added to the buffer
    - Number of steps to take on each training set batch
    - Number of steps to take on replay buffer samples, per training set batch
    - Step size for training set batch updates
    - Step size for replay buffer updates

The first baseline allows us to verify our method can learn non-linear representations of images. Further, comparison against CNNs trained with varying sizes of replay buffer gives an indication of how much information our model can fuse via filtering.

| Layer # | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Layer type | Conv. | Max pool | Dense | Softmax |
| Num. filters | 16 | - | - | - |
| Kernel size | $5 \times 5$ | $2 \times 2$ | - | - |
| Dense num. inputs | - | - | 2304 | - |
| Dense num. outputs | - | - | 10 | - |
| Inc. bias | ✓ | - | ✓ | - |
| Activation $g(\cdot)$ | Leaky ReLU | Linear | Linear | - |

Table 4.5: The baseline CNN architecture.

**Hyperparameter Tuning** Hyperparameters for all models were tuned on validation sets generated by randomly subsampling 15% of the training set. As mentioned above, we evaluate all methods on different training set sizes. We tune the hyperparameters for the baseline models separately for each training set size. Conversely, with GBP Learning we use the same hyperparameters for all dataset sizes.

The full specification of the convolution factor graph is given in Table 4.4, and the corresponding CNN baseline model in Table 4.5.

#### 4.3.3.1 Results

The test accuracies as a function of dataset size are presented in Figure 4.10. In the low data regime, GBP Learning comprehensively outperforms all baselines, likely due to regularising priors and marginalising over uncertain activations. With the full training set, GBP Learning achieves a test accuracy of $98.16 \pm 0.03\%$, significantly higher than the linear classifier. Further, we outperform most CNN + Adam variants except those with large replay buffers (3600 examples/6% of training data, 6000 examples/10% training data), to which we perform similarly. We are thus encouraged that our model can learn complex relationships incrementally, consolidating new examples into the parameter posterior online.

Full dataset training with GBP Learning takes $\sim 3$ hours on NVIDIA RTX3090 GPU. While this is much slower than the few minutes to train the CNNs on CPU, we note that the software to train NNs with backpropagation has benefited from decades of optimisation, as have well-established GPU hardware platforms. In contrast, our current GBP Learning implementation runs on a hardware and software stack optimised for DL. We believe orders of magnitude efficiency could be gained by developing on a more tailored setup with optimised software and hardware with on-chip memory. See further discussion in Section 4.5, and the

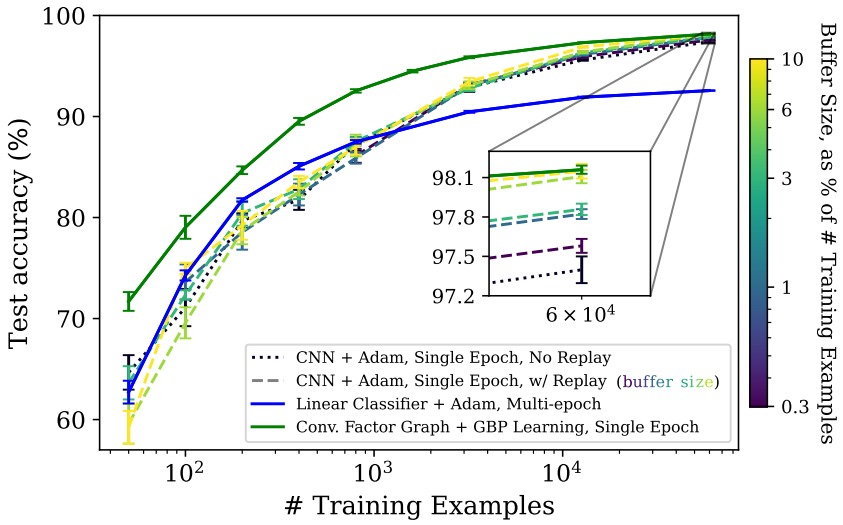



Figure 4.10: **Single epoch MNIST results.** GBP Learning outperforms other methods in the small data regime, and performs similarly to a CNN with a replay buffer of $6 \times 10^3$ examples on the full training set. Error bars cover $\pm$1SE over 5 seeds.

experiments in Sections 4.3.3.3 and 4.3.3.4 which imply significant speedups may be possible in model-parallel settings.

#### 4.3.3.2 Dependence on Number of Iterations

Next, we characterise the properties of GBP Learning from an adaptive computation perspective: i.e. what is the trade-off between the amount of computation (in GBP iterations) and the predictive accuracy. We train replicas of the convolutional factor graph model (summarised in Table 4.4) on MNIST, each with a different number of GBP iterations per training batch. We then evaluate the test accuracy of each of the trained models multiple times: for differing numbers of test-time GBP iterations. All configurations used a batch size of 50 at train time and 200 at test time. We ran our experiments with $50, 100, 200, 400, 800, 1600$ iterations per batch at both train time and test time.

The results are presented in Figure 4.11. Accuracy is monotonic in both the number of training and the number of testing iterations, with the greatest accuracy achieved at 1600 iterations (98.2%). However, good test performance can be achieved with relatively few GBP iterations per training batch ($\sim 200$), as long as a sufficient number of iterations is run at test time ($\geqslant 200$, $\sim 98\%$). We believe this result demonstrates potential of GBP

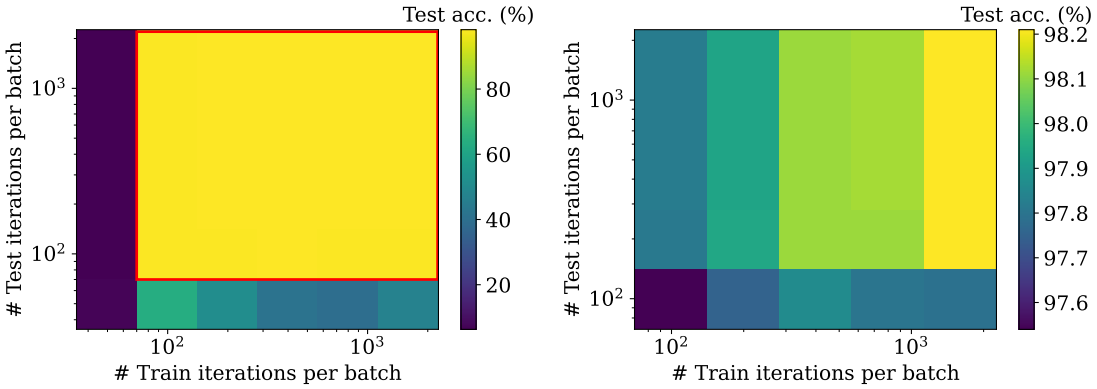



Figure 4.11: **Dependence of MNIST test accuracy on the number of GBP iterations at train and test time**. The right-hand plot covers the range of iterations marked by the red box in the left-hand plot, with the colours rescaled accordingly.

Learning in settings where speed-accuracy trade-offs are important: one could learn a good model or generate a decent prediction quickly, which could then be refined if additional time/compute were available.

#### 4.3.3.3 Asynchronous Training

In decentralised computation settings, it is usually not possible to synchronise message updates due to the required communication. The above experiments were performed with ordered forward-backward message sweeps which are not applicable in this setting, so we seek to understand the performance of GBP Learning instead with random message ordering. We thus train the same convolutional factor graph above (Table 4.4) in a layer-wise asynchronous manner with random ordering. At each iteration, we uniformly sample a sequence of $L = 4$ layers with replacement to determine the layer update order. Sampling with replacement means some layers may not be updated at all during an iteration, where some may be updated multiple times. Other than the message schedule, all other experimental settings are left unchanged.

Layerwise-random training yields a final accuracy of $98.11 \pm 0.04\%$ ($\pm$1SE over 5 seeds), close to $98.16 \pm 0.03\%$ achieved by the same model trained with ordered forward/backward sweeps. Further, we see that the evolution of test accuracy over the course of training (Figure 4.12) is similar in both cases. This result implies that GBP Learning is robust to inter-layer message scheduling, and can work well in scenarios with no global synchronisation.

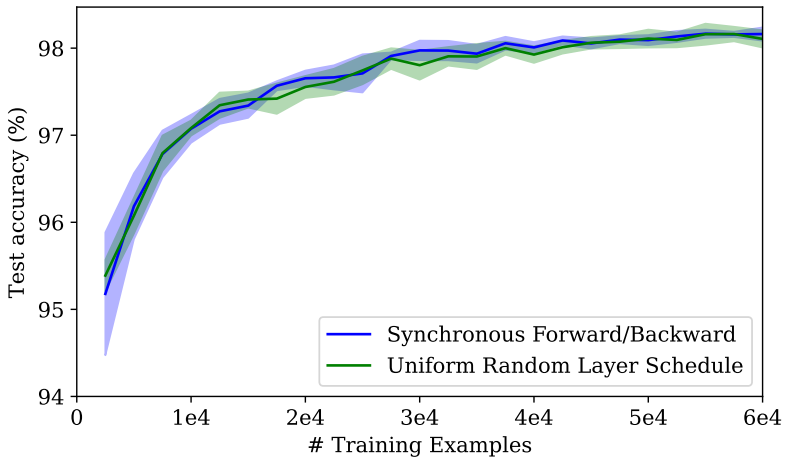



Figure 4.12: **MNIST results with layerwise-random update schedules.** The evolution in test accuracy over the course of training, for a convolutional factor graph trained on MNIST, with i) synchronous forward/backward sweeps and ii) random layer ordering. Intervals represent ±1SE either side of the mean, over 5 random seeds. Both set ups present similar progressions in performance during training. Error bars cover 1SE either side of the mean over 5 random seeds.

#### 4.3.3.4 Model-Parallel Training

As message updates in GBP depend only on the local state of the graph, they can be executed in parallel for different parts of the model. This can potentially accelerate GBP Learning, reducing wall-clock time by parallelising inference over multiple processors. However, such parallelism involves a trade-off, between the staleness of the input signals used to compute messages and the frequency of inter-processor communication which may be slow and requires pausing computation. We now empirically analyse this trade-off for GBP Learning in a model-parallel setting.

**Setup** For ease of implementation, we simulate model-parallelism with a message schedule which alternates between: i) running multiple message updates for the layers with even indices, ii) multiple message updates for layers with odd indices. This simulates model-parallelism because all the updates that happen in each phase are independent – the updates in layer 0 do not affect those in layer 2 during phase i) for example. Thus the messages are identical to those which would be computed when placing the layers on different processors without communication.

When evaluating the effect of parallelism on performance, we control for total computation. We do so by running the same total number of message updates in all cases, but vary how those updates are divided between local computation in parallel, and sequential computation with inter-layer messages. We use a total of 500 message updates per layer per training batch, which is divided up as e.g. 500 repeats of layer schedule $[4, 2, 1, 3]$, 250 repeats of $[4, 4, 2, 2, 1, 1, 3, 3]$, 100 repeats of $[4, 4, 4, 4, 4, 2, 2, \ldots, 3]$ and so on. We test such configurations with $1, 2, 5, 10, 50, 100, 250$ and 500 repeated local iterations per communication. Note that in the final case, there is only a single round of inter layer communication, after all 500 local updates have been executed. Other than the message schedule, we retain the same experimental set up as above (Table 4.4) — including the test-time message schedule.

**Results** We plot the results in Figure 4.13. Unsurprisingly, the extreme case in which all updates happen in isolation with one round of communication performs poorly (bright yellow line, bottom of Figure 4.13a) — similarly to randomly guessing the label. However, we see with relatively few rounds of inter-layer communication, we are able to achieve performance similar to communication every update. For example, a fairly extreme setting with only 5 rounds of 100 local iterations results in a test accuracy of 98.01%, close to 500 rounds of 1 local iteration (98.22%).

We are thus encouraged that GBP Learning can perform well in a highly model-parallel setting, and converge to good solutions with infrequent inter-processor communication. This implies a genuinely distributed implementation could reduce the wall-clock time for training significantly.

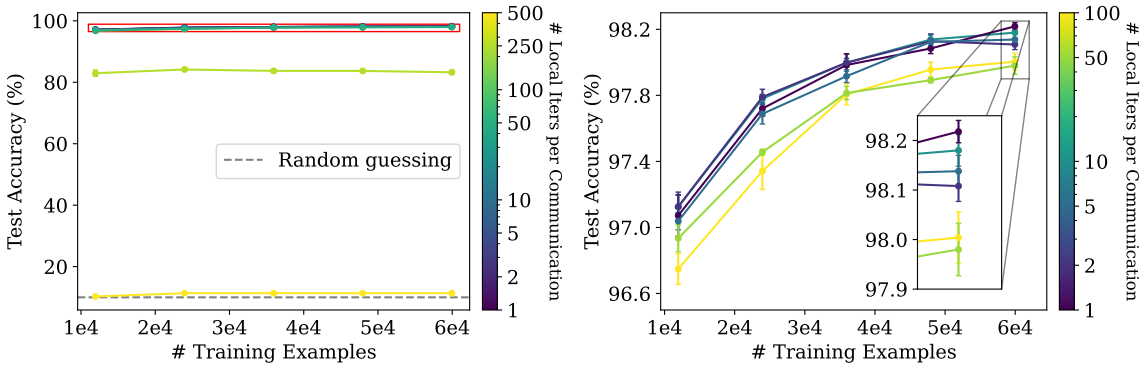



Figure 4.13: **Model-parallel training with GBP Learning**. We train the convolutional factor graph of Table 4.4 on MNIST with model-parallel simulation, finding that a large fraction of the message updates can be executed in parallel without significant loss of performance. The region inside the red rectangle in (a) is expanded in (b) (with different colour scale).

#### 4.3.3.5 Comparison with Lucibello et al. [2022]

After the development of GBP Learning, we were made aware of the work of Lucibello et al. [2022], done independently of ours. Their work is similar, in that they train NN-like models with message passing algorithms, however their method is only designed for MLP-like models, and they must train for $\sim 100s$ of epochs to achieve good performance. See Section 4.4 for further discussion.

To understand how the performance of our method compares, we evaluate on the same image classification benchmarks presented in Lucibello et al. [2022]: MNIST, FashionMNIST and CIFAR-10. We test the convolutional model described in Table 4.4 trained with GBP Learning, and contrast the results with those from the dense factor graphs in Lucibello et al. [2022]. For GBP Learning, we retain the same architecture and hyperparameters used for MNIST — we do not conduct any task-specific tuning.

The results are given in Table 4.6. Both approaches perform similarly for FashionMNIST, but our method achieves higher accuracy for both MNIST and CIFAR-10. In the latter case, we outperform Lucibello et al. [2022] by more than 10%. Moreover, we obtain such performance with incremental training, passing over the training set once only, where Lucibello et al. [2022] train for 100 epochs.

We note a significant difference in the relative improvement between FashionMNIST

| Dataset | Lucibello et al. [2022] | GBP Learning |
|---|---|---|
| MNIST | $97.40 \pm 0.04$ | $\mathbf{98.16 \pm 0.03}$ |
| FashionMNIST | $88.2 \pm 0.1$ | $88.2 \pm 0.1$ |
| CIFAR-10 | $41.3 \pm 0.1$ | $\mathbf{53.1 \pm 0.3}$ |

Table 4.6: **Comparison of image classification test accuracies (%) between GBP Learning and Lucibello et al. [2022]**. Ranges cover 1SE either side of the mean over 5 seeds.

(0%) and CIFAR-10 (11.8%), which we conjecture may result from CIFAR-10 benefiting more from translation equivariance of the convolutional layer. In FashionMNIST, the objects of interest are mostly centred and fill the frame, where in CIFAR-10 the objects are smaller and occur in different regions of the images.

### 4.3.4 Example Application: Combining Learning and Control

We now describe how our method enables the combination of hand-designed and learnable components. Specifically we combine learning and control into a single inference procedure. We will first describe the model, and how the control problem is encoded, before applying it to a simple task and presenting the results. Our method is an example of control-as-inference [Levine, 2018], where the control signals are random variables in a graphical model which is conditioned to satisfy the objective of the control problem.

**Setup** We create a hand-designed factor graph representation of the control problem (see Figure 4.14). However, we relax the dynamics factors (which make up the chain between state variables) to be learnable rather than fixed. Thus we have a graphical model in which environment dynamics, future states $\mathbf{s}_{t+1}, \ldots, \mathbf{s}_T$ and future actions $a_{t+1}, \ldots, a_T$ are unknown latent variable. We parameterise the dynamics model with network parameters $\theta$ which are included as variables in the factor graph. For simplicity, we do not include activations as additional variables, but include the $\theta$-parameterised network in the factors directly. Our dynamics factors, which connect a $(\mathbf{s}_t, a_t)$ pair to the next state $\mathbf{s}_{t+1}$ and dynamics model parameters $\theta$, have energy

$$E_{\text{dyn}}\left(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}, \theta\right) = \frac{\left\|\mathbf{s}_{t+1} - f\left(\mathbf{s}_t, a_t; \theta\right)\right\|_2^2}{\sigma_{\text{dyn}}^2}, \tag{4.24}$$

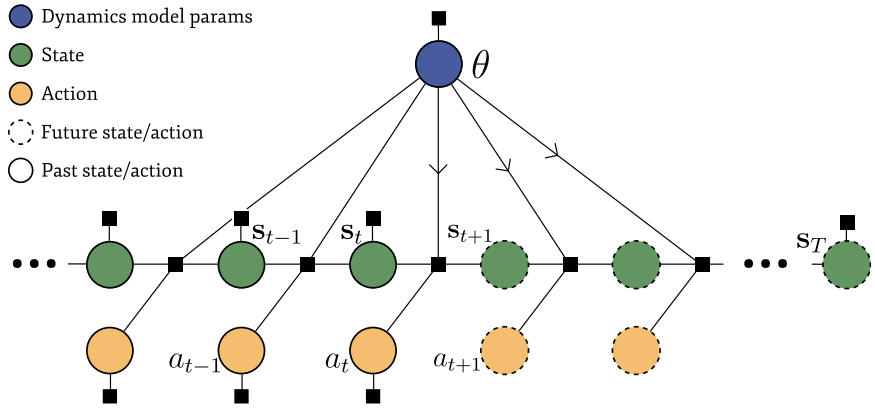



Figure 4.14: **An example factor graph for joint learning and control.** Previous states and actions are observed, and thus are connected to unary observation factors, while future states and actions are not in general. However, goal states are encoded as unary factors on the corresponding state variables (in this case for the final state $\mathbf{s}_T$). Arrows on edges indicate GBP messages along these edges are only passed in one direction.

where $f(\cdot, \cdot; \theta)$ is the dynamics model network. These factors enforce pairs of states, together with the control signal, to be consistent with the learnt dynamics model.

We note however, that this relationship is bidirectional. The dynamics factors encourage both the state variables to be consistent with the dynamics model *and* the dynamics model weights to be consistent with state variables. The latter is not desirable when we consider that the factor graph contains both previously observed states and unknown future states. In particular, we do not want to learn dynamics from future state variables which are free to take on any value. To this end, we apply a crude modification to the GBP inference procedure: we block messages from future dynamics factors to the parameters $\theta$, only allowing messages from variable to factor along these edges (see arrowed edges in Figure 4.14).

It may be unclear at this point how this model enables control. The trick is to condition future state variable(s) to take the value of a goal state $\mathbf{s}_G$, which is considered a solution to the control problem. If we then run inference on $a_{t+1}, \dots, a_{T-1}$, we can generate a *plan*, i.e. work out what actions should be taken to achieve the specified goal state given the current dynamics model. We do this in a model predictive control routine, i.e. at each step we

1. Take an action according to our current plan $a_t = \text{argmax}_{a_t} p(a_t | \mathbf{s}_T = \mathbf{s}_G, \mathbf{s}_1, \dots, \mathbf{s}_t)$.
2. Observe the next state $\mathbf{s}_{t+1}$ after executing $a_t$. Add the corresponding observation factor to the model.

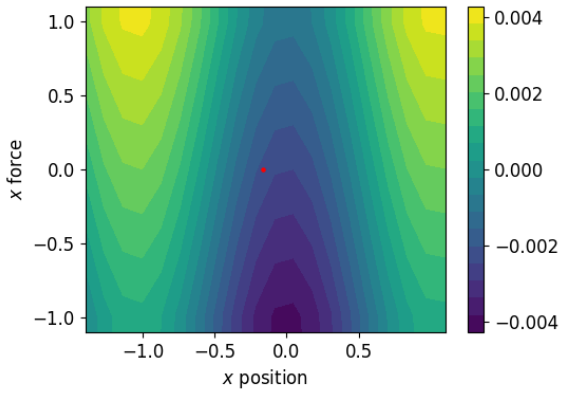



Figure 4.15: **Ground truth dynamics of the mountain car environment.** The contour plot shows how a give 1D location (horizontal axis) and applied force (vertical axis) map to a resultant acceleration (colour-coded).

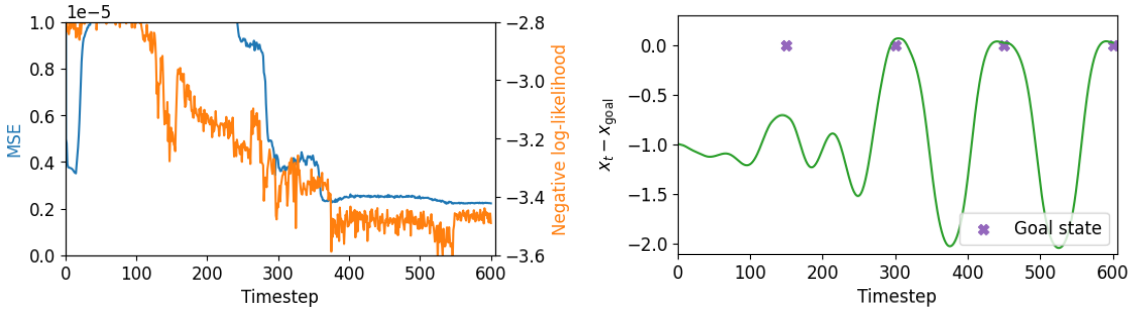
3. Run GBP in the updated model to get new dynamics parameters $p(\theta)$ and plan $p(a_{t+1}, \ldots, a_{T-1}, \mathbf{s}_{t+2}, \ldots, \mathbf{s}_T | \mathbf{s}_T = \mathbf{s}_G, \mathbf{s}_1, \ldots, \mathbf{s}_{t+1})$.

#### 4.3.4.1 Mountain Car Example

We apply the proposed framework to the mountain car problem [Moore, 1990], a common reinforcement learning test bed. In this simple one-dimensional simulation, a car is placed near the bottom of the valley situated between two hills, and the objective is to reach the top of one of the hills (the goal state). The acceleration of the car is limited such that it cannot reach the goal state greedily from the valley floor, by applying maximum acceleration. Instead it must build momentum by oscillating back and forth between the two hills until reaching the goal state. We use the continuous mountain car implementation in the `Gymnasium` library[4]. The state at each timestep is a position-velocity tuple $\mathbf{s}_t = (x_t, v_t)$ and the action $a_t \in [-1, 1]$ is the applied 1D force. The dynamics factors in the model

As a dynamics model, we use a single hidden layer MLP, with 8 hidden units and Leaky-ReLU activations. It takes a state-action pair as input and predicts the resulting acceleration. As the underlying dynamics of the simulation are known (plotted in Figure 4.15), we can compare our learnt model with ground truth to assess accuracy. We run the simulation for 600 steps, and enforce the goal state target every 150 steps, i.e. the agent should attempt to reach the top of the hill every 150 steps. If the goal state was only enforced on the final

[4] https://github.com/Farama-Foundation/Gymnasium




(a) Evolution of model accuracy

(b) Trajectory of the car

Figure 4.16: **Quantitative results for the mountain car simulation**. The accuracy of the dynamics model improves as more observations are added (a). The car trajectory (b) shows that the car misses the goal state at $t = 150$, but has a sufficiently accurate model to achieve it at $t \in \{300, 450, 600\}$.

state, we found the agent did not explore sufficiently to learn a good dynamics model, and was unable to reach the goal with one attempt.

**Results** The planned states and learnt model at timesteps $2, 100, 150, 300, 570$ are shown in Figure 4.17. We see that to begin with ($t = 2$), the learnt model is inaccurate (see ground truth in Figure 4.15), and the planned states are highly uncertain. As the car makes more observations around the bottom of the valley the dynamics model becomes locally accurate ($t = 100$), but is still not sufficient to reach the hilltop at $t = 150$. This failed attempt, however, provides observations in previously unseen regions, which improves the model enough to navigate to the hilltop at $t = 300$. The model continues to improve and by $t = 570$ we see it resembles ground truth and results in sensible plans.

These observations are in accordance with the quantitative results. We show the accuracy of the dynamics model (evaluated on a dense grid over the domain plotted in Figure 4.15), over the 600 timesteps in Figure 4.16a and the displacement of the car relative to the goal location in Figure 4.16b. These plots show that our method is both i) able to learn the dynamics of the environment from observations and ii) infer good control signals to achieve the goal.

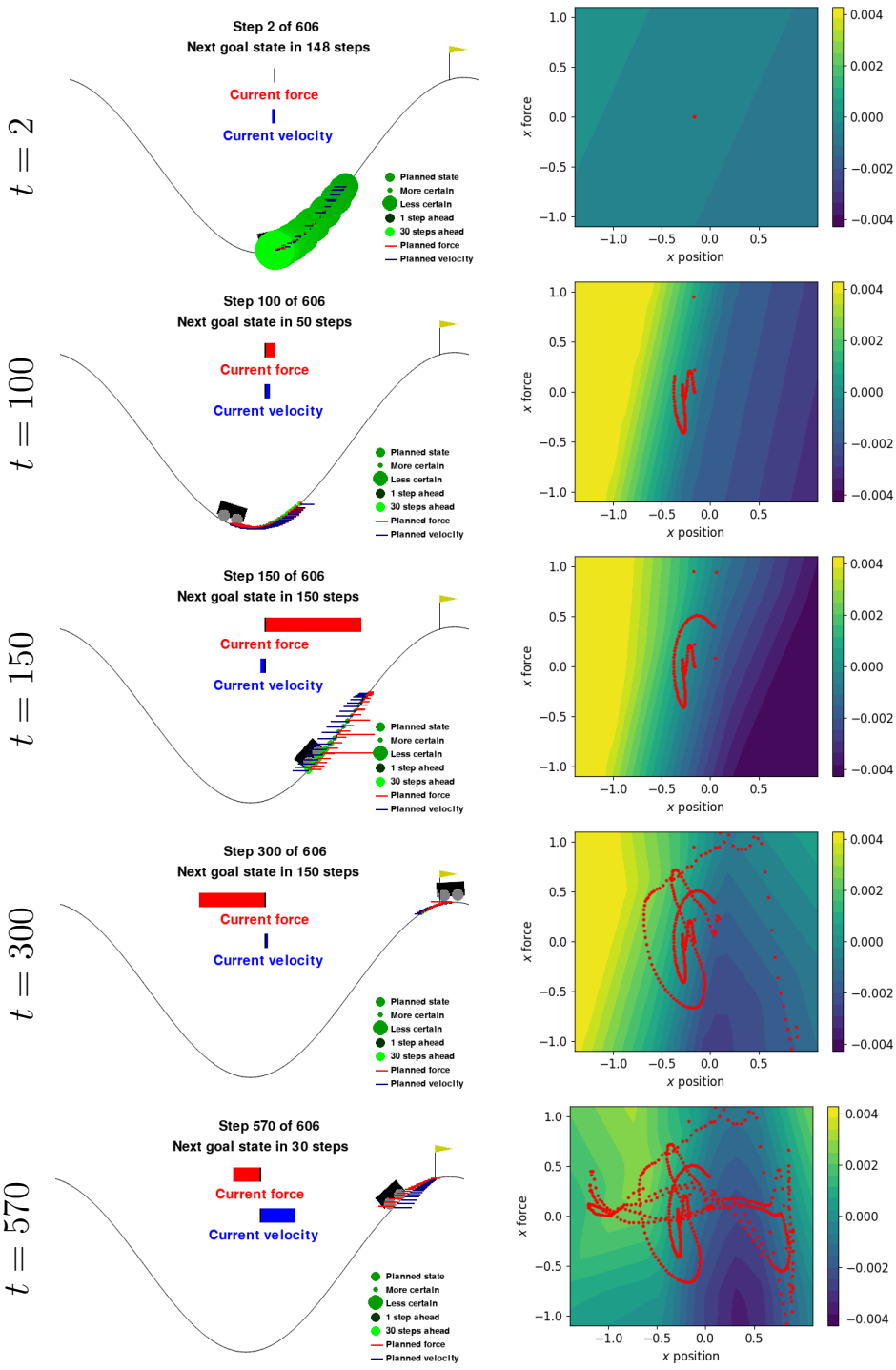



Figure 4.17: **Results of joint learning and control on the mountain car environment.** The left column shows the position of the car and the planned states over time (see key in lower right-hand corner of plots). The goal state location is marked with the flag. The The right column shows the learnt dynamics model (ground truth in Figure 4.15), where the vertical axis is applied force $a_t$ and the horizontal is position $x_t$. Red points in the right-hand plots mark previously observed $(x_t, a_t)$ pairs. Best viewed digitally.

## 4.4 Related Work

Our models can be viewed as probabilistic energy-based models (EBMs) [LeCun et al., 2006, Du and Mordatch, 2019] whose energy functions are the sum of the quadratic energies for all factors in the graph. The benefit of using a Gaussian factor graph is a model which normalises in closed form without resorting to expensive MCMC sampling. While the quadratic energy may seem constraining, we add capacity via non-linear factors and overparameterisation. For most EBMs, parameters are attributes of the factors, where we include them as variables in our graph. As a result, learning and prediction are the same procedure in our approach, but two distinct stages for other EBMs.

Of the EBM family, restricted Boltzmann machines RBMs [Smolensky, 1986] are of particular relevance. Their factor graph resembles a single layer, fully connected version of our model, however RBMs are models over discrete variables which are unable to capture the statistics of continuous natural images. While exponential family generalisations exist, such as Gaussian-Bernoulli RBMs [Welling et al., 2004], scaling RBMs to multiple layers remains a challenging problem. To our knowledge there are no working examples of training stacked RBMs jointly, instead they are trained greedily, with each layer learning to reconstruct the activations of the trained and fixed previous layer [Hinton et al., 2006, Hinton and Salakhutdinov, 2006]. As an artefact of this, later layers use capacity to learn artificial correlations induced by the early layers, rather than correlations present in the data. Global alignment is then found by fine tuning with either backprop [Hinton and Salakhutdinov, 2006] or wake-sleep [Hinton et al., 2006]. We have found GBP Learning to work in multiple layer models without issue.

Our approach also relates to Bayesian DL [Neal, 2012] as we seek to infer a distribution over parameters of a deep network. Common methods to train Bayesian NNs (BNNs) are based on e.g. variational inference [Blundell et al., 2015, Gal and Ghahramani, 2016], the Laplace approximation [MacKay, 1992, Ritter et al., 2018], Hamiltonian Monte Carlo [Neal, 2012], Langevin-MCMC [Zhang et al., 2019]. Each of these comes with benefits and drawbacks, but we note two distinguishing features of our method. Computationally, prior Bayesian DL methods rely on backprop, and so inherit its restrictions to distributed and asynchronous training. In contrast, BP inference is stateful and based on local update rules, enabling distributed and asynchronous GBP Learning. Further, our activations are random variables, allowing us to model (and resolve) disagreement between bottom-up and top-down signals.

The factor graphs we consider relate to multi-layer predictive coding (PC) models [Millidge et al., 2022, Rao and Ballard, 1999, Friston, 2005, Buckley et al., 2017, Alonso et al., 2022]. Designed as a hierarchical model of biological neurons in the brain, multi-layer PC networks are trained by minimising layerwise-local, Gaussian prediction errors similar to our inter-layer consistency factors. Like ours, they also include likelihood factors which ensure observed input/output variables remain close to the observation value. Moreover, some works have noted the suitability of PC for layerwise-parallelism and emerging hardware platforms [Salvatori et al., 2023]. Our framework provides a new approach to train PC models with GBP, an alternative to the standard approach using gradient descent of a variational free energy [Millidge et al., 2022]. The work of Parr et al. [2019] is an exception, and uses BP for inference in a model of biological neurons. However, they assume the model parameters are known, where we infer parameters jointly with activations. Moreover, we consider larger scale machine learning applications.

Much work has focused on augmenting BP with DL [Nachmani et al., 2016, Satorras and Welling, 2021, Yoon et al., 2019, Lázaro-Gredilla et al., 2021]. In contrast, our method does DL *within* a BP framework. George et al. [2017] propose a hybrid vision system in which visual features and graph structure are learnt in a separate process to the discrete BP routine used to parse scenes (given features and graph). The work of Lázaro-Gredilla et al. [2016] is more similar to ours in that parameters are treated as variables in the graphical model and updated with BP. However, their factor graph comprises only binary variables, making it ill-suited to natural images. In addition, max-product BP is used to find a MAP solution, and not a marginal posterior estimate. This precludes incremental learning via Bayesian filtering.

Most relevant to our work is that of Lucibello et al. [2022], who use GBP to train factor graphs similar to MLPs. Their work was not known to us during the development of GBP Learning, and was only brought to our attention after. We note their focus is somewhat distinct from ours. We are primarily interested in learning with GBP for its decentralised, asynchronous computational properties, where Lucibello et al. [2022] are focused on adapting GBP for "mini-batch training on GPUs" and showing that "approximate Bayesian predictions [e.g. with Bayesian model average (1.20)] have higher accuracy than pointwise ones".

Their method relies on analytically derived message updates via moment matching which only apply to dense architectures, and they focus on models with binary weights and sign activations. In contrast, our approach is straightforward to apply to any network structure without rederivation of messages, assuming the appropriate factor energies can be specified.

Further, we focus on continuous weights in direct analogy to NN parameters. Though they similarly minibatch by filtering over parameters, they visit each datapoint multiple times and run a small number of GBP iterations during each visit. After so few iterations, the resulting messages are unlikely to constitute an accurate posterior, and ad-hoc "forgetting" factors are necessary to avoid overcounting data seen multiple times. In contrast, we train with only a single epoch, running GBP to convergence on each batch, enabling straightforward filtering.

## 4.5 Conclusion

We have introduced GBP Learning, a method for learning in Gaussian factor graphs. Parameters are included as variables in the graph and learnt using the same BP inference procedure used to estimate all other latent variables. Inter-layer factors encourage representations to be locally consistent, and multiple layers can be stacked in order to learn richer abstractions.

Experimentally, we have shown a shallow, learnable model improves over a hand-crafted method on a video denoising task, with further performance gains coming from stacking multiple layers. We have also trained convolutional factor graphs for image classification with GBP Learning. On MNIST, we demonstrate encouraging sample efficiency, and reach comparable single epoch performance to an equivalent CNN with a replay buffer of $6 \times 10^3$ examples. GBP Learning outperforms Lucibello et al. [2022] by 11.8% on CIFAR-10 and 0.8% on MNIST. Furthermore, we have shown GBP Learning can be executed in a highly model-parallel manner, with infrequent communication, as well as with layerwise-random message schedules. This implies it would be robust to genuinely distributed, asynchronous execution. Last, we showed how using factor graph inference for parameter learning provides a means to combine hand-designed and learnable modules in the same model, with a simple control example.

**Scaling up** While we find these results encouraging, we highlight scaling up GBP Learning to bigger, more complex models and datasets as an exciting future direction. Our current implementation is built on software and run on hardware heavily optimised for DL. Scaling up GBP Learning will require a bespoke hardware/software system, which can better leverage the distributed nature of GBP inference. In particular, we believe processors with memory local to the compute cores [Graphcore, Cerebras] to be a promising platform for BP. Different sections of the factor graph could be mapped to different cores. Local message

updates, between factors and variables on the same core, would be cheap and could be updated at high frequency, with occasional inter-core communication to ensure alignment between different parts of the model (cf layerwise-parallel experiments in Section 4.3.3.4). Low-level GBP primitives, written in e.g. Poplar[5], could be used to ensure the message updates make best use of high-bandwidth, local memory. A similar system was applied to bundle adjustment problems with GBP [Ortiz et al., 2020], and was found to be around 24× faster on IPU [Graphcore] than a state-of-the-art CPU solver.

**Model design** As we discussed in Section 4.2, our proposed approach to GBP Learning constitutes one of many possibilities, and there remains a large design space to explore. We highlight three potential directions, but recognise there are many others.

First, we note that we have only considered GBP for marginal inference of scalar variables. This is potentially restrictive and likely prevents capturing the rich correlation structure if the energy landscape of deep networks. By including multidimensional variables in the factor graph, we would expect to mitigate this, and possibly enable more stable BP due to a reduced number of loops in the model. As higher dimensional variables incur greater computational cost per iteration, only variables which are likely to be highly correlated, e.g. the activations or weights within a layer, should be combined.

Second, we have included all activations as random variables. While this maximises the potential for layerwise distribution of computation, it may make inference more challenging in deep networks due to large numbers of tight loops. Instead, we could use factors between sets of activation variables which comprise multiple NN layers. For example, the energy for a double-layer factor could be,

$$E_{\text{l,2layer}}\left(\mathbf{h}_l, \mathbf{h}_{l+2}, \theta_l, \theta_{l+1}\right) = \frac{\left\|\mathbf{h}_{l+2} - \mathbf{f}_{l+1}\left(\mathbf{f}_l\left(\mathbf{h}_l; \theta_l\right), \theta_{l+1}\right)\right\|_2^2}{\sigma_l^2} \tag{4.25}$$

where the parameters for both layers $\theta_l, \theta_{l+1}$ are random variables, but the activations in the intermediate layer $l+1$ are not. This framework would make possible a spectrum of models, with full GBP Learning at one end and standard, deterministic NNs at the other.

**Generative/discriminative hybrid modelling** Last, we consider model directionality. Factor graphs are undirected joint models over their variables. We should therefore be able to answer different probabilistic "queries" — i.e. what are the values of variables $a, b, c$ given

[5] graphcore.ai/products/poplar

observed values of $x, y, z$? — by running inference in the same trained model with different observed nodes. Thus, deep factor graphs should be bidirectional: we should be able to add observation factors which clamp any subset of input and output nodes, run BP and infer a sensible posterior over the unobserved nodes.

While we have demonstrated the potential of GBP Learning on generative tasks (e.g. video denoising, Section 4.3.2.1) and discriminative tasks (e.g. image classification, Section 4.3.3), our exploratory attempts to build hybrid, bidirectional models were unsuccessful. We found that the form of the inter-layer factors implies a preference of direction, even if the model is technically undirected. In particular, we found an MLP-like factor graph with feedforward factors such as

$$E_{\text{discrim.}}\left(\mathbf{h}_{l-1}, \mathbf{h}_l, \theta_l\right) = \frac{\left\|\mathbf{h}_l - g\left(W_l^\intercal \mathbf{h}_{l-1} + \mathbf{d}_l\right)\right\|_2^2}{2\sigma_l^2} \ , \tag{4.26}$$

was able to e.g. predict a semantic class but could not inpaint a masked region of an image, and vice versa for factors with energy functions preferring the generative direction, such as

$$E_{\text{gener.}}\left(\mathbf{h}_{l-1}, \mathbf{h}_l, \theta_l\right) = \frac{\left\|\mathbf{h}_{l-1} - g\left(W_l^\intercal \mathbf{h}_l + \mathbf{d}_l\right)\right\|_2^2}{2\sigma_l^2} \ . \tag{4.27}$$

We therefore postulate that the design of symmetric factor energies, which do not imply a preferred direction, may enable better hybrid models.

# Chapter 5

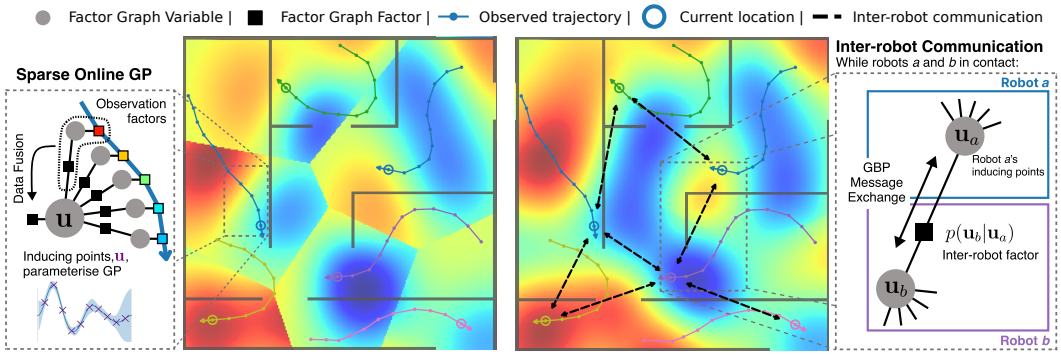
# A Distributed Gaussian Process Model for Multi-Robot Mapping

We now detail the work from our paper *A Distributed Gaussian Process Model for Multi-Robot Mapping*, under review at *IEEE International Conference on Robotics and Automation, 2026* [Nabarro et al., 2025].

In this chapter, we propose a multi-robot learning method for collaborative inference of a global function using only local experience and computation. We present a sparse Gaussian process (GP) model with a factorisation that mirrors the multi-robot structure of the task, and admits distributed training via Gaussian belief propagation (GBP). Our loopy model outperforms Tree-structured Gaussian processes (TSGPs) [Bui and Turner, 2014] and can be trained online and in settings with dynamic connectivity. We show that such distributed, asynchronous training can reach the same performance as a centralised, batch-trained model, albeit with slower convergence. Last, we compare to DiNNO [Yu et al., 2022], a distributed neural network (NN) optimiser, and find our method achieves superior accuracy, is more robust to sparse communication and is better able to learn continually without experience replay.

## 5.1 Introduction

Many mapping tasks involve large areas and dynamic maps, suggesting a divide-and-conquer approach using multiple robots may be beneficial. Centralised communication and computation are often not feasible in domains like environmental monitoring [Dunbabin and




(a) Before inter-robot exchange (b) After inter-robot exchange

Figure 5.1: **Distributed GP mapping.** We propose a factor graph approach to distributed mapping. Each robot maintains a block of inducing points which parameterise their local GP (far left), into which observations can be fused online. Communicating robots harmonise their models by connecting inducing points with GP consistency factors and exchanging GBP messages (far right). In this example, the map value at each point is predicted by the closest robot. Before communication (centre left) the map is discontinuous and inaccurate, but inter-robot communication resolves discontinuities using only local communication.

Marques, 2012], robotic agriculture [Albani et al., 2017] or space exploration [Yliniemi et al., 2014], which involve many robots accumulating large datasets in poorly connected regions. Further, centralised approaches can be slow where robots may need up-to-date local models to guide exploration. These considerations motivate decentralised approaches, with local computation and mesh connectivity.

We present a multi-robot algorithm to map an arbitrary global function using only local observation, computation and peer-to-peer communication. We use sparse GPs with inducing points which approximate a full GP at lower computational cost, and act as local summaries of the fitted function (see e.g. orange bars in Figure 5.2). They provide a useful abstraction for data fusion and inter-robot exchange. Further, they are assumed to be Gaussian distributed, enabling decentralised inference with GBP.

We first assess the TSGP model of Bui and Turner [2014], which partitions inducing points into sparsely connected blocks. Each block covers a distinct region of the input space and is maintained by a single robot. Robots exchange GBP messages with their neighbours to enforce consistency between their inducing points. By design, TSGP restricts inter-robot connectivity to form a tree, which guarantees fast convergence. However, we show empirically that this constraint limits accuracy: inducing points close in Euclidean

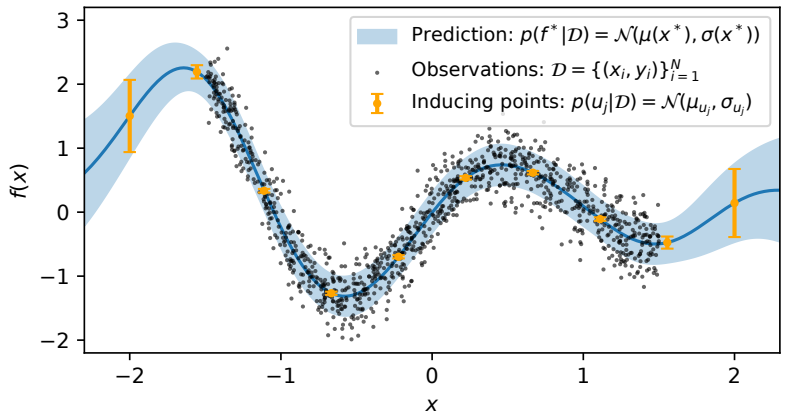



Figure 5.2: **A 1D FITC GP example.** The inducing points (orange) capture the important characteristics of the function with less redundancy than the original observations (black). Inducing points are random variables. The errorbars and shaded regions cover one standard deviation either side of the mean.

space may not be directly connected in the model, causing discontinuities between their inducing points. Further, the drawback of the tree-constraint is most apparent in dynamic multi-robot scenarios — when two robots meet, they may be unable to share information if doing so would introduce a cycle.

To address this limitation, we propose a simple innovation: *relax the tree constraint to allow loops*. While GBP in our model no longer converges in two sweeps, we demonstrate a substantial improvement in accuracy. Moreover, in settings with ever-changing inter-robot distances, and therefore dynamic connectivity, TSGP's message schedule, requiring fixed, coordinated connectivity to build a tree, is not feasible. In a loopy model, however, new edges can be added asynchronously, as and when pairs of robots make contact with one another. Thus, our loopy model naturally exploits all robot encounters, and we show it can converge with asynchronous message passing. Our proposed loopy factor graph approach is shown in (Figure 5.1).

There is substantial prior work on multi-robot learning. In particular, Yu et al. [2022] introduce a distributed approach to NN training (DiNNO) using alternating direction method of multipliers (ADMM) [Boyd et al., 2011]. We believe our method has many advantages over DiNNO. First, inter-robot consistency is only enforced in regions of input space where both robots have inducing points. In contrast, DiNNO enforces consensus in

NN weight-space, which may affect the function globally in unobserved regions. Second, our probabilistic model supports on-the-fly observation fusion for efficient online learning, where DiNNO must store and replay observations due to catastrophic forgetting [French, 1999]. Last, each robot only stores its own inducing point block, a local map-segment, where each DiNNO robot must store a global map estimate, limiting scalability.

We summarise our contributions as follows:

1. A new distributed GP model, trainable with GBP, which outperforms TSGPs [Bui and Turner, 2014].
2. Empirical analysis showing the model can be built on-the-fly and admits distributed, asynchronous inference via GBP. It is thus well-suited to multi-robot mapping.
3. Experimental evaluation of our method on environmental monitoring and occupancy mapping tasks. We empirically demonstrate superior performance to DiNNO [Yu et al., 2022].

## 5.2 Background

As noted in our brief description of GPs (Section 2.5), they are bottlenecked by the inversion of the covariance matrix $K_x \in \mathbb{R}^{N\times N}$ in (2.95), which has complexity $\mathcal{O}(N^3)$. This is prohibitive for large datasets, motivating approximations with cheaper training and prediction.

### 5.2.1 Inducing Point Approximations

Inducing point GPs seek to compress the dataset of $N$ observations into a set of $M < N$ pseudo-datapoints (inducing points) which capture its important characteristics at a reduced computational cost. One such method is the Fully Independent Training Conditional (FITC) approximation [Snelson and Ghahramani, 2005, Csató and Opper, 2002]. The inducing points, $\mathbf{u} \in \mathbb{R}^M$, are latent variables which approximate $f(\cdot)$ at the chosen set of input locations $\zeta = [\mathbf{z}_1^\intercal, \dots, \mathbf{z}_M^\intercal]^\intercal$. The approximating generative model is

$$q(\mathbf{f}, \mathbf{u}) = q(\mathbf{f}|\mathbf{u})p(\mathbf{u}) = \prod_{i=1}^{N} p(f_i|\mathbf{u})p(\mathbf{u}), \tag{5.1}$$

where the conditional independence of $\mathbf{f}|\mathbf{u}$ is a model assumption which enables cheaper inference and learning, $p(\mathbf{u})$ is a dense GP prior over the inducing points. The resulting posterior predictive $p\left(f(\mathbf{x}^*)|\mathbf{y}, X\right)$ requires inversion of dense $M \times M$ and diagonal $N \times N$ matrices, thus avoiding the $\mathcal{O}(N^3)$ cost of inverting a dense $N \times N$ matrix. We present an example of a fitted FITC GP approximation in Figure 5.2.

### 5.2.2 Tree-Structured Gaussian Process Approximations

While inducing point approximations such as FITC are efficient when $M << N$, Bui and Turner [2014] argue that many datasets require $M \approx N$ for accurate modelling. For efficient inference in this setting, they propose approximating a dense prior over $\mathbf{u}$ by dividing them into blocks $\mathbf{u} = [\mathbf{u}_{B_1}^\intercal, \ldots, \mathbf{u}_{B_K}^\intercal]^\intercal$ and connecting them in a tree

$$q(\mathbf{u}) = q(\mathbf{u}_{B_1}) \prod_{k=2}^{K} q(\mathbf{u}_{B_k}|\mathbf{u}_{\mathrm{par}(B_k)}), \tag{5.2}$$

where $B_1$ is the root node and $\mathrm{par}(B_k)$ is the parent of $B_k$.

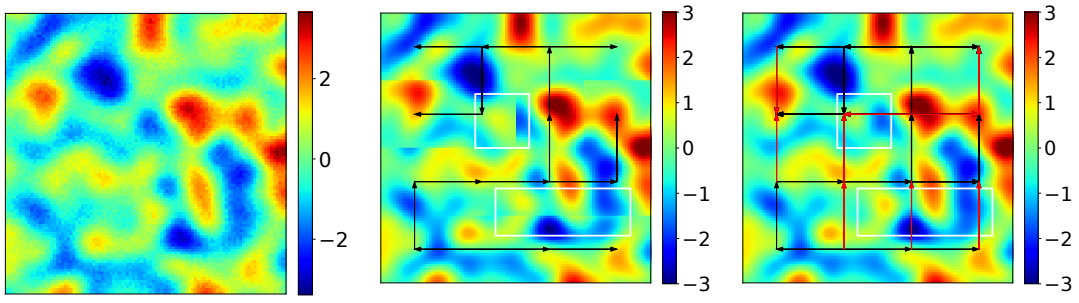
Each block generates a disjoint subset of observations $\mathbf{f}$

$$q(\mathbf{f}|\mathbf{u}) = \prod_{k=1}^{K} q(\mathbf{f}_{C_k}|\mathbf{u}_{B_k}). \tag{5.3}$$

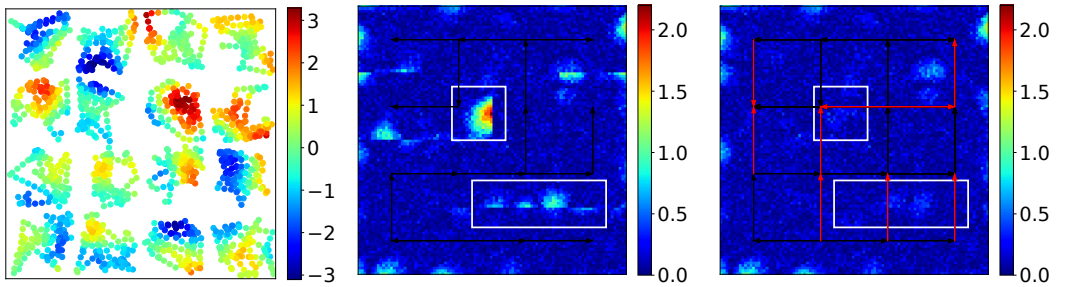
with connections between elements of $\mathbf{f}$ and $\mathbf{u}$ determined by their proximity. For example, $f_a := f(\mathbf{x}_a)$ may be connected to the $\mathbf{u}_{B_k}$ whose input locations have a centroid closest to $\mathbf{x}_a$. By minimising the KL divergence from the dense joint $p(\mathbf{f}, \mathbf{u})$ to their approximating generative model $q(\mathbf{f}, \mathbf{u})$, they derive the form of factors $q(\mathbf{u}_{B_k}|\mathbf{u}_{\mathrm{par}(B_k)})$ and $q(\mathbf{f}_{C_k}|\mathbf{u}_{B_k})$, showing they should be equal to standard GP conditionals $p(\mathbf{u}_{B_k}|\mathbf{u}_{\mathrm{par}(B_k)})$ and $p(\mathbf{f}_{C_k}|\mathbf{u}_{B_k})$.

As TSGP has a sparse set of factors $\{p(\mathbf{u}_{B_k}|\mathbf{u}_{\mathrm{par}(B_k)})\}_k$ in place of a dense prior $p(\mathbf{u})$, inference can be efficient in models where the number of inducing points is large. Bui and Turner [2014] show how the posteriors $\{q(\mathbf{u}_{B_k}|\mathbf{y})\}_k$ can be inferred with GBP message passing. The tree-structured topology ensures GBP converges in one sweep from leaves to root and another from root to leaves.

After inference, predictions at a new $\mathbf{x}^*$ can be made by finding a nearby block of inducing points $\mathbf{u}_{B_{k^*}}$ (e.g. with input locations whose centroid is closest) and estimating



(a) Ground truth + noise (b) TSGP (c) TSGP + 9 Edges



(d) Training data (e) TSGP Abs. Error (f) TSGP + 9 Edges Abs. Error

Figure 5.3: **Limitations of TSGP in 2D.** Some robots remain unconnected to ensure tree connectivity (as in b). This leads to discontinuities in the predicted function at the boundary between unconnected robots (e), which can be reduced significantly by adding edges which form loops (f) (see areas highlighted with white rectangles).

$f(\mathbf{x}^*)$ by marginalising over the posterior $q\left(\mathbf{u}_{B_{k^*}}|\mathbf{y}\right)$:

$$q\left(f(\mathbf{x}^*)|\mathbf{y}\right) = \int p\left(f(\mathbf{x}^*)|\mathbf{u}_{B_{k^*}}\right) q\left(\mathbf{u}_{B_{k^*}}|\mathbf{y}\right) d\mathbf{u}_{B_{k^*}}. \tag{5.4}$$

This prediction relies only on the closest $\mathbf{u}_{B_{k^*}}$, avoiding test-time communication with other parts of the graph.

## 5.3 Method

### 5.3.1 Improving Over Tree-Structured GPs

The following features TSGP make it a good candidate for distributed multi-robot learning:

1. Each inducing block $\mathbf{u}_{B_k}$ models a different subset of observations (5.3), so robots

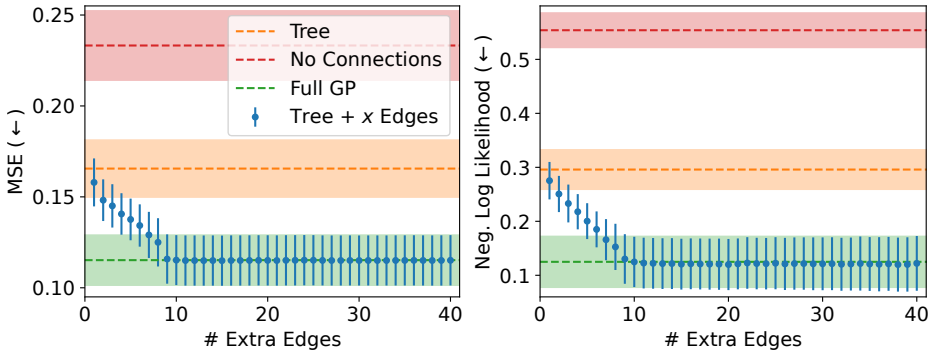



Figure 5.4: **Additional edges improve the prediction accuracy of TSGP.** The prediction error (vertical axis) reduces as more additional edges are added to the TSGP model (horizontal axis, $x = 0$ is TSGP), but plateaus after around 9 additional edges, at which the performance is equal to a dense GP prior on all inducing variables (green). All results averaged over 10 repeats, error bars cover ±1 standard error (SE) around the mean.

can update their local posteriors $q(\mathbf{u}_{B_k}|\mathbf{y}_{C_k})$ with new experience without needing to communicate with others.

2. Each $\mathbf{u}_{B_k}$ only connects to its neighbours (5.2). With GBP inference, this ensures inter-robot messages are local exchanges — no global communication is needed.

**Evaluating TSGP** To evaluate of TSGP, we train it on simulated multi-robot data. We sample a static, ground truth function from a GP over $(x, y)$, which the robots observe (with additive Gaussian noise) at each timestep (Figure 5.3a). The input space covering $[-1, 1]\times[-1, 1]$ is divided into a 4×4 grid, with a robot occupying each cell. We generate robot trajectories by randomly sampling a target within each cell, which the robot heads towards at a constant speed. Once within $\epsilon = 0.02$, the target is resampled. The observations are presented in Figure 5.3d. Each robot's inducing points are placed on a regular grid spanning its cell. Blocks of inducing points between robots in adjacent cells are connected, such that the graph of all inducing point blocks forms a tree (black arrows in Figure 5.3b). The model is batch-trained on the full dataset.

The resulting predictions and error are shown in Figs. 5.3b and 5.3e. The error is concentrated at boundaries where between cells which are not connected, as a consequence of the tree structure. A tree which connects nodes on a regular grid in 2 or more dimensions is guaranteed to result in unconnected neighbouring nodes. The fitted function suffers from a lack of inter-robot smoothing factors across these boundaries.

To quantify this effect, we train a sequence of models, starting at TSGP and adding edges one-by-one, prioritising connections between robots which are close in $(x, y)$ space but distant in the graph. For context, we also train a model without inter-robot connections (each robot fits an independent inducing point GP), and a model with a dense GP prior over the inducing points of all robots. We train each model 10 times on different datasets, each randomly sampled as in Figure 5.3d. We evaluate on a dense grid, but remove test points close to (less than $d = 0.06$ from) any training observations.

The results (Figure 5.4) show that accuracy improves as edges are added, until $\sim 9$ additional edges, after which the performance matches the dense GP prior. Examining the predictions and errors for a model with 9 extra edges on the example dataset (Figs. 5.3c and f) we see the discontinuities are significantly reduced (e.g. inside white rectangles in Figs. 5.3e and 5.3f). Further, at 9 extra edges (red arrows in Figure 5.3c), each node connects to its immediate NESW neighbours and longer-range edges have little impact. Indeed, this is consistent with repeated analyses on $3 \times 3$ and $5 \times 5$ grids, where plateaus in accuracy occur at 4 and 16 extra edges respectively.

### 5.3.2 Asynchronous Model Construction and Training

We have shown that we can improve distributed mapping performance by adding more inter-robot connections to TSGP, but we assumed that all data were available upfront, and any robot could be connected to any other. In most multi-robot mapping applications these assumptions are not realistic, so here we design a model which can be trained incrementally, with dynamic connectivity depending on the robots' relative locations. We first describe how each robot updates its local model with new observations, then discuss where new inducing points are added, before outlining how robots should connect their inducing points upon meeting.

#### 5.3.2.1 Adding and Removing Observations

Desirable features for a robot's local model include i) scalability w.r.t the number of observations, ii) ability to incorporate new observations efficiently, iii) ability to discard observation after their contribution to the inducing points has been accounted for. To this end, we employ the FITC approximation [Snelson and Ghahramani, 2005, Csató and Opper, 2002] between inducing points and observations (see factor graph in Figure 5.1a). For a robot with inducing points $\mathbf{u}_{B_i}$ and observations $\mathbf{y}_{C_i} = [y_1^{(C_i)}, \ldots, y_{N_i}^{(C_i)}]^\intercal$ at $X^{(C_i)} =$

$[(\mathbf{x}_1^{(C_i)})^\intercal, \ldots, (\mathbf{x}_{N_i}^{(C_i)})^\intercal]^\intercal$, this approximation can be written as

$$q(\mathbf{y}_{C_i}, \mathbf{f}_{C_i} | \mathbf{u}_{B_i}) = \prod_{n=1}^{N_i} p(y_n^{(C_i)} | f_n^{(C_i)}) p(f_n^{(C_i)} | \mathbf{u}_{B_i}). \tag{5.5}$$

For $N_i < M_i$, this approximation ensures the complexity of inference is $\mathcal{O}\left(M_i^2 N_i\right)$, and therefore scalable to large $N_i$. Further, a new observation $y_{N_i+1}^{(C_i)}$ can be easily added online, by creating new $p(y_{N_i+1}^{(C_i)} | f_{N_i+1}^{(C_i)})$, $p(f_{N_i+1}^{(C_i)} | \mathbf{u}_{B_i})$ factors, a $f_{N_i+1}^{(C_i)}$ variable and calculating the resulting message to $\mathbf{u}_{B_i}$. Last, individual observations can be discarded after their message from factor $p(f_{N_i+1}^{(C_i)} | \mathbf{u}_{B_i})$ to $\mathbf{u}_{B_i}$ has been computed. We maintain a unary factor on $\mathbf{u}_{B_i}$ which accumulates the messages from discarded observations (see LHS of Section 5.1a for illustration). After this, factors $p(y_{i,m} | f_{i,m})$, $p(f_{i,m} | \mathbf{u}_i)$ and variable $f_{i,m}$ are removed.

Discarding observation factors online introduces two sources of approximation relative to batch fitting. First, in most cases we add inducing points incrementally, as the robot gathers experience. Inducing points which have been added after an observation factor has been discarded will not benefit from the corresponding message. This does not affect inference when the inducing point locations are fixed throughout. Second, for non-linear observation factors $p(y_n^{(C_i)} | f_n^{(C_i)})$, the message to $\mathbf{u}_{B_i}$ depends on the estimate of $f_n^{(C_i)}$. The estimate of an $f_n^{(C_i)}$ variable added after observation $y_j^{(C_i)}$ was removed could have been affected by the removed observation, thus the message from $f_n^{(C_i)}$ to $\mathbf{u}_{B_i}$ is an approximation.

#### 5.3.2.2 Adding New Inducing Points

We efficiently add inducing points to the model using the *greedy variance selection* method of Burt et al. [2020]. This method i) chooses an inducing point at the observation location with the highest marginal output variance under $p(f_n^{(C_i)} | \mathbf{u}_{B_i})$, ii) adds it to inducing point locations $\zeta_{B_i}$, iii) updates the model and iv) repeats until $M_i$ is reached.

In our setting, the set of observations grows with time; we do not have the static dataset assumed in Burt et al. [2020]. Nonetheless we employ greedy variance selection in an online fashion, adding a new inducing point to the model every $k$ timesteps. We restrict our choice of new locations to observations which have not yet been fused and discarded (see Section 5.3.2.1). This ensures that the new inducing point will benefit from a strong message sent by the observation factor at a co-located observation.

In addition to selecting inducing points from observations, we also find that adding

inducing points close to the boundary between two connecting robots provides the model with more flexibility to make their representations consistent. We use two strategies for doing this: i) each robot adding a line of inducing points along their shared boundary; or ii) each robot adding inducing points at the same locations as a subset of the other's, near their shared boundary.

#### 5.3.2.3 Adding New Connections

We next consider how two communicating robots should exchange information to synchronise their representations. We connect their inducing points via a shared factor, but what form should this factor take? We first observe that the optimal prior over inducing points, i.e. that which most closely approximates a full GP over observations, is itself a full GP over the inducing points:

$$p(\mathbf{u}_{B_1}, \dots, \mathbf{u}_{B_b}) = \mathcal{N}(\mathbf{0}, K_{\zeta_{\text{all}}}) \tag{5.6}$$

where $K_{\zeta_{\text{all}}} \in \mathbb{R}^{M \times M}$ is the kernel matrix over all inducing points. This joint prior can be factorised sequentially:

$$p(\mathbf{u}_{B_1}, \dots, \mathbf{u}_{B_b}) = p(\mathbf{u}_{B_1}) \prod_{i=2}^{b} p(\mathbf{u}_{B_i} | \{\mathbf{u}_{B_j}\}_{j<i}) \tag{5.7}$$

meaning it can be distributed s.t. each robot "owns" one factor. However, this fully connected structure is computationally infeasible: inference scales as $\mathcal{O}(M^3)$, where $M := \sum_i \dim(\mathbf{u}_{B_i})$. Moreover, in realistic multi-robot settings, each robot can only communicate with nearby peers, where this factorisation requires all-to-all communication.

Despite these constraints, (5.7) provides a theoretical target: it implies an inter-robot communication protocol which would converge to the ideal $p(\mathbf{u}_{B_1}, \dots, \mathbf{u}_{B_b})$ in the limit of full connectivity. The protocol is as follows:

1. **Initialisation:** To begin, each robot has a unary GP prior on its inducing variables $p(\mathbf{u}_{B_i}) = \mathcal{N}(\mathbf{0}, K_{\zeta_{B_i}})$, equal to the message that $\mathbf{u}_{B_i}$ would receive from a global dense prior over inducing points (5.6), due to the marginal consistency of GPs. This ensures message updates are well-conditioned, enabling inference in each robot's local model before any inter-robot contact.

2. **Inter-robot exchange:** Suppose robot $i$ now encounters $j$. The two compare their indices to determine which will act as the parent in the connecting factor. If $i > j$, for example, robot $j$ will be the parent whose inducing points will condition $i$'s prior. Three updates are now executed: i) the unary prior is amended to a conditional: $p(\mathbf{u}_{B_i}) \rightarrow p(\mathbf{u}_{B_i}|\mathbf{u}_{B_j})$ and connected to $\mathbf{u}_{B_j}$, ii) variable-to-factor messages $m_{\mathbf{u}_{B_i} \rightarrow p(\mathbf{u}_{B_i}|\mathbf{u}_{B_j})}$, $m_{\mathbf{u}_{B_j} \rightarrow p(\mathbf{u}_{B_i}|\mathbf{u}_{B_j})}$ are updated, and iii) factor-to-variable messages $m_{p(\mathbf{u}_{B_i}|\mathbf{u}_{B_j}) \rightarrow \mathbf{u}_{B_i}}$, $m_{p(\mathbf{u}_{B_i}|\mathbf{u}_{B_j}) \rightarrow \mathbf{u}_{B_j}}$ are updated.

3. **Decoupling:** The message exchanges in 2 are executed every timestep the two robots are within range. After separation, the child $i$ stores the previous state of $m_{\mathbf{u}_{B_j} \rightarrow p(\mathbf{u}_{B_i}|\mathbf{u}_{B_j})}$ and $p(\mathbf{u}_{B_i}|\mathbf{u}_{B_j})$, which may be further conditioned on another robot's inducing variables later. A unary factor with value $m_{p(\mathbf{u}_{B_i}|\mathbf{u}_{B_j}) \rightarrow \mathbf{u}_{B_j}}$ is added to $\mathbf{u}_{B_j}$.

In the limit of all-to-all communication, we highlight that this routine constitutes inference in a model with a full GP prior (5.7). However, there are two sources of approximation in our approach. First, robots which are always far away and never make contact may never be connected via a shared edge. These robots are treated as independent. Second, robots which connect and then move out of range will be propagating stale messages. Thus, the asynchronous nature of the connections may limit performance. We empirically evaluate the impact of these approximations in Section 5.3.3.

#### 5.3.2.4 Distributed Hyperparameter Learning

Kernel hyperparameters $\theta$ are typically tuned by gradient ascent of the log-marginal likelihood (see Section 2.5). Bui and Turner [2014] propose a message-passing routine to update kernel hyperparameters in TSGP. However, their method requires a full up-down sweeps of the tree to compute the gradient before it is applied. This is not applicable in our case, with cyclic models and sporadic connectivity. Instead, we compute stochastic, approximate marginal likelihood gradients based on each robot's individual experience, and use this to update their local copy of the kernel hyperparameters. We interleave hyperparameter tuning and GBP, updating the GP factors with the latest hyperparameters at each stage.

#### 5.3.2.5 Prediction

Prediction at a test point $\mathbf{x}^*$ can be made directly from the posterior of a chosen block of inducing points as in (5.4). The choice of which inducing point posterior $q(\mathbf{u}_{B_i}|\mathbf{y})$ is used to

predict $\mathbf{x}^*$ should ideally be done according to how observations are connected to inducing points during training. However, this may not be possible in practice, and the mapping from test points to robots will ultimately depend on details of the application and where $q(\mathbf{u}_{B_i}|\mathbf{y})$ are stored at prediction time. In our experiments we explore two methods for mapping test queries to robots: i) predict at $\mathbf{x}^*$ with the robot whose current location $\mathbf{x}_{r_i,t}$ is closest, $r^* = \text{argmin}_{r_i \in \{r_1,\dots,r_R\}} ||\mathbf{x}^* - \mathbf{x}_{r_i,t}||^2$ and ii) pre-allocate regions $\{\mathcal{R}_i\}$ of the input space in which each robot is responsible for mapping and predicting: $r^* = r_i \iff \mathbf{x}^* \in \mathcal{R}_i$.

### 5.3.3 Evaluation of Asynchronous Model Construction

To quantify the impact of asynchronous model construction and GBP (as described in Section 5.3.2), we compare its performance to a batch-fitted model with dense inducing point prior. We generate the trajectories and observations as in Section 5.3.1, and again place inducing points on a regular grid within each robot's cell. However, in this case the robots learn online, incorporating new observations, connecting and exchanging messages as per Section 5.3.2. Robots can only communicate with others in a $d_{\text{comm}} = 0.15$ radius of their current location. This distance is a small fraction of the overall input space covering $[-1, 1] \times [-1, 1]$. For comparison, we fit the batch models every 400 steps, using all observations made so far. We evaluate predictions in six randomly chosen $0.3 \times 0.3$ subregions which are "held out": robots may move through these regions but do not make observations when inside them.

The results in Figure 5.5 show that while the distributed implementation (blue) is slower to converge, it achieves the same accuracy as the batch-fitted dense GP prior over all inducing points (green). Further, the distributed implementation is able to rapidly achieve performance superior to a batch-fitted TSGP (orange) and a batch fitted model without connections (red). These results suggest we can maintain accuracy while learning a map in a decentralised, asynchronous manner.

## 5.4 Experiments

We now evaluate our method in practice, testing it on two distributed mapping problems: sea-surface temperature prediction and occupancy mapping. In both instances, we compare against DiNNO [Yu et al., 2022]. When presenting results, we denote our method *DistGP*.

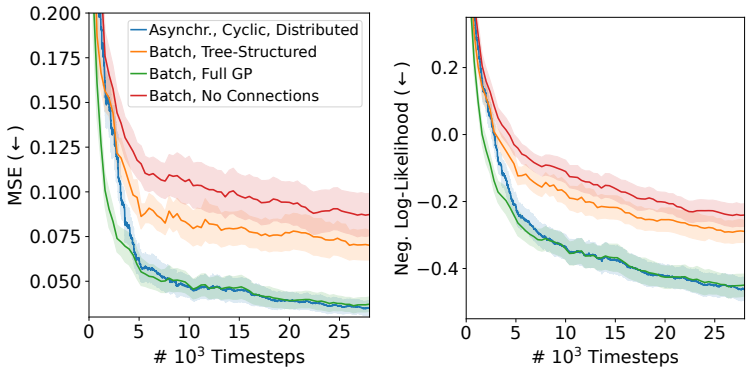



Figure 5.5: **Asynchronous + distributed vs batch + centralised model.** We compare i) online learning with ad-hoc inter-robot connections and ad-hoc GBP (blue) against ii) a centralised implementation with synchronous GBP sweeps on batch data (red, orange, green). The former converges to the same performance as the centralised, dense prior (green). Averages over 10 seeds, error bars cover ±1 SE around the mean.

### 5.4.1 Dynamic Sea Surface Temperature Tracking

Many environmental monitoring applications require dynamic maps over large areas, in regions with poor connectivity. These challenges motivate a decentralised multi-robot mapping approach like ours. To evaluate our method in this setting, we simulate a multi-robot sea surface temperature (SST) monitoring problem.

**Setup** The optimally interpolated sea surface temperature (OISST) dataset [Reynolds et al., 2007, Huang et al., 2021] combines satellite, buoy and ship measurements, and interpolates them on a dense grid of 1/4 lat/lon resolution at daily frequency. We select 5 disjoint ocean regions, each covering a 4400×4400km$^2$ area, and divided into a $5 \times 5$ grid. A fleet of 25 aquatic robots is spread over the region, one per cell, which make pointwise SST measurements as they move around. We assume the robots move at a low speed of 7km/h, can communicate with others within $d_{\text{comm}} = 1500$km and observe the temperature every 3 hours. As we only have access to daily mean temperatures, we assume the field remains constant throughout each day.

The aim is for the combined robot predictions over the area to accurately match the ground truth map at any given point in time. As the robots move slowly and the map evolves with time, their observation coverage is limited. To do well, therefore, requires robots are able to exchange useful information with their neighbours to improve their predictions

| Region | DistGP (Ours) | DiNNO |
|---|---|---|
| Central Atlantic | $\mathbf{0.134 \pm 0.001}$ | $0.175 \pm 0.001$ |
| North Indian Ocean | $\mathbf{0.160 \pm 0.001}$ | $0.222 \pm 0.001$ |
| South Indian Ocean | $\mathbf{0.230 \pm 0.001}$ | $0.316 \pm 0.002$ |
| West Pacific | $\mathbf{0.108 \pm 0.001}$ | $0.166 \pm 0.001$ |
| East Pacific | $\mathbf{0.240 \pm 0.002}$ | $0.303 \pm 0.001$ |

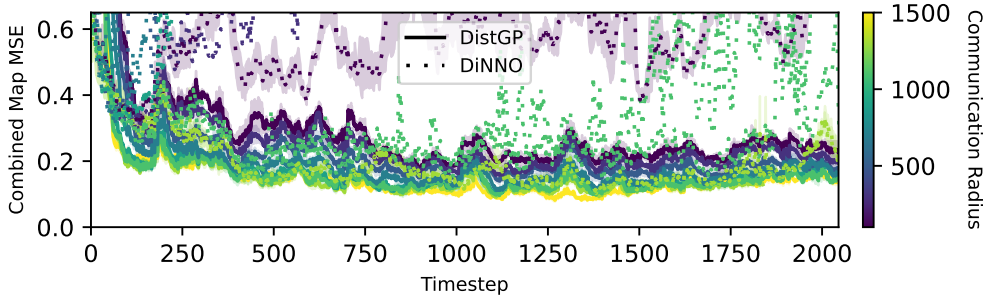
Table 5.1: **SST prediction MSE.** Error in predicted temperatures over the map, averaged over all timesteps after a 25 day "burn-in". Intervals cover ± SE over 4 seeds.

in less recently observed regions. Combined predictions are generated on a dense grid, with each robot predicting the SST at all test points inside their cell.

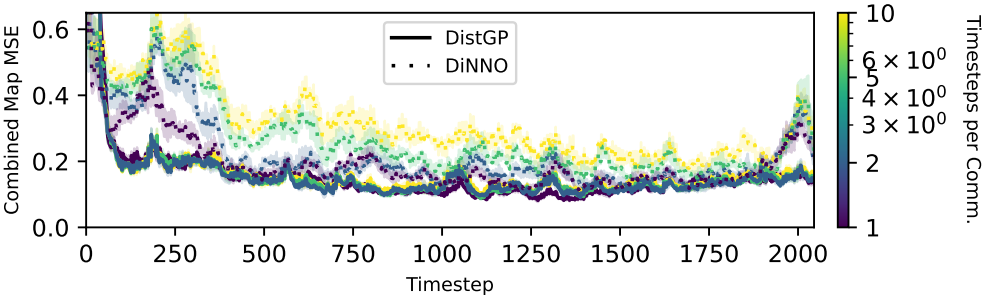
We train our method by adding observations online and incorporating their messages into the inducing point posterior as per Section 5.3.2. Each robot retains 5 observations in their model after which we fuse the observation factors into a unary factor on the inducing points. Every other step, one of these observations is selected as a new inducing point via greedy variance selection. While this amounts to a relatively dense set of inducing points, we only retain the 200 most recent inducing points, as older inducing points become irrelevant to current predictions as the field evolves. When two robots in adjacent cells connect, they both add a line of inducing points close to the shared border to provide extra flexibility to make their fitted functions consistent. We use the product of two Matérn-1/2 kernels as our covariance function, one over 2D space and one over time; and we tune the kernel hyperparameters as well as the observation noise with the distributed method described in Section 5.3.2.4.

The base model for DiNNO is a 6-layer MLP with width 64 and leaky-ReLU activations. At each timestep we train these networks using minibatch samples from the previous two weeks worth of observations. During hyperparameter tuning, we found this gave rise to more accurate prediction than training on data sampled from all previous observations.

For better observation coverage of their designated regions, each robot follows the following simple routine for path planning. They first create a regular grid of points within their cell. These are the candidate locations for the next target location. The minimum distance from each cell point to the trajectory of the 100 most recent robot locations is computed. The next goal location is then uniformly sampled from the three grid points which are furthest from any of the previous trajectory points. The robot then navigates directly to this location before replanning. This routine i) ensures the region is covered, and




(a) Differing communication radius (colour-coded)




(b) Differing interval between communication steps (colour-coded)

Figure 5.6: **Ablation of different communication regimes.** We see that DistGP is more robust to infrequent communication than DiNNO.

visited locations are not revisited soon after, and ii) has sufficient stochasticity to prevent oscillations between a few points in the region. We follow this routine for both robots. While we could in theory use the predictive uncertainty of the GP for active learning, we are interested in how the distributed learning methods compare given similar observations, where different planning algorithms would result in different data distributions.

Hyperparameters for both methods were chosen based on simulations in the Central Atlantic region from 2022/01/01 to 2022/07/01. We then execute online training in simulations between 2022/07/01 and 2023/03/15, evaluating the accuracy of the global map on a dense grid every timestep after a 25 day "burn in" period.

**Results** The quantitative results of average map accuracy (Table 5.1) show that DistGP significantly outperforms DiNNO for all regions, suggesting it enables more efficient learning

and communication. Further, we present examples of map predictions in Figure 5.7. Both methods appear to reach reasonable consensus among robots in their predicted map. However, the DistGP predictions are better able to capture finer details in the map where DiNNO often appears over-smoothed. We postulate this may be due to over-regularisation caused by enforced consensus between NN weights, where constraining nearby inducing points to be similar is a local rather than global constraint, ensuring most of the map is unaffected and details can be retained.

We further examine the robustness of both methods under different communication settings, varying the frequency of inter-robot communication and the communication radius of each robot in the Atlantic region (Figure 5.6). The results show that DistGP is far more robust to less frequent communication and shorter communication radii than DiNNO.

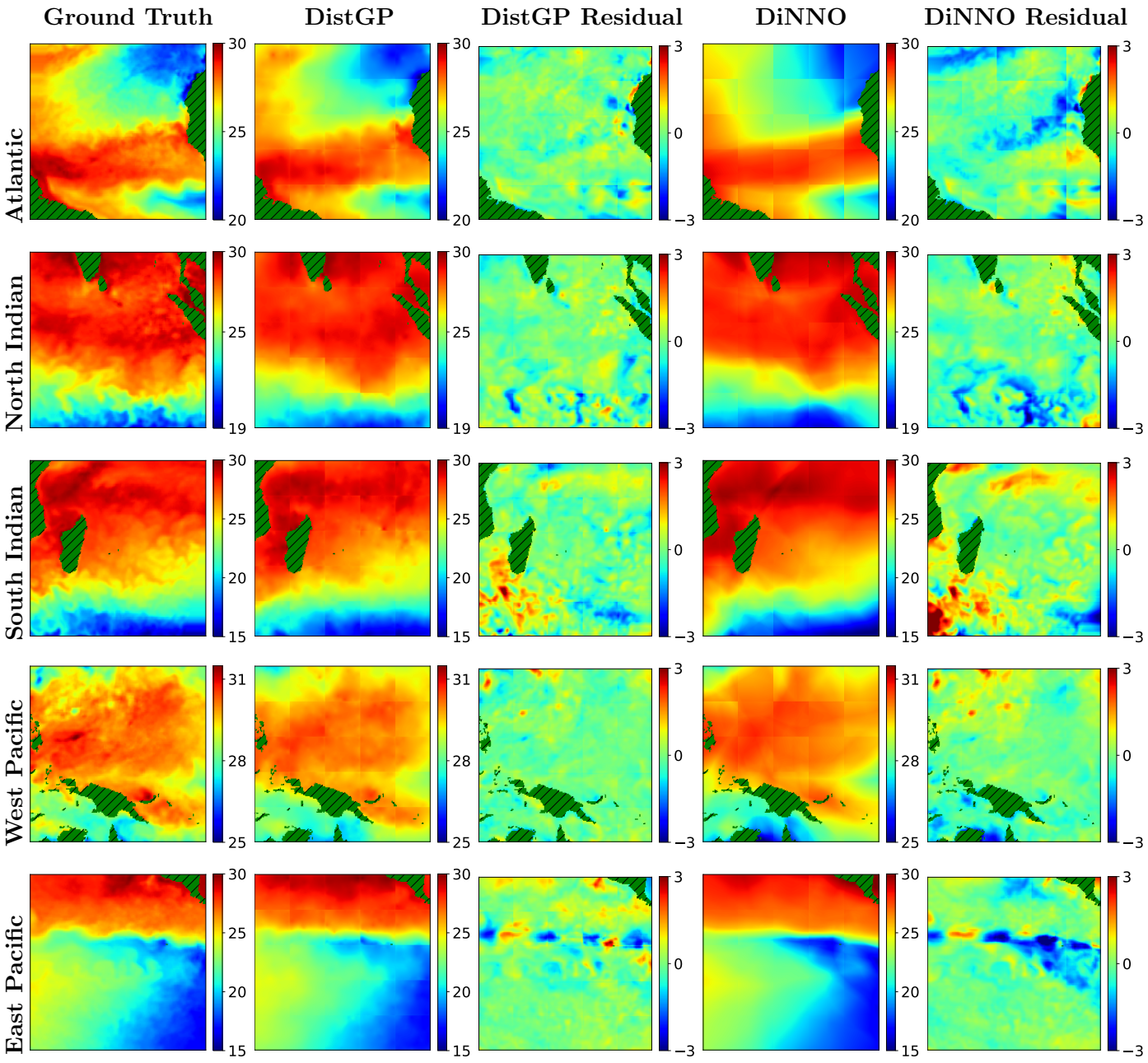



Figure 5.7: **Example OISST maps learnt by DistGP and DiNNO.** Columns correspond to different predictions/residuals, rows correspond to different ocean regions. We see that DistGP is better able to capture high spatial frequencies, and misses fewer important variations in the map.

### 5.4.2 Online Occupancy Mapping

We also evaluate on the 2D online mapping task from Yu et al. [2022]. The trajectories of 7 robots (Figure 5.8b) around a 2D environment (Figure 5.8a) are predefined, and each robot makes a sweep of simulated LIDAR measurements from their position at each timestep (Figure 5.8c). The aim is to use these observations, and inter-robot communication, to learn a map online.

**Setup** To simulate streaming data, Yu et al. [2022] give each robot a buffer of the 400 most recent scans to train its model (total scans per trajectory $\sim 4 \times 10^3$). However, each robot has to repeat its trajectory $\sim$ 100s of times to achieve an accurate final map. We believe this is due to catastrophic forgetting in NNs, and therefore ask whether our probabilistic representation enables greater efficiency.

We treat each LIDAR scan of 500 observations as a batch, use it to update the inducing point posterior, then discard it as per Section 5.3.2.1 (in contrast to DiNNO's 400 scan buffer). As the occupancy observations are discrete $y_n \in \{0, 1\}$, we use binary observation factors with energy equal to binary cross-entropy (BCE), and the sigmoid-transformed GP prediction $p_n = \sigma(f(\mathbf{x}_n))$ as the predicted probability. We linearise these factors, approximating them as Gaussian to allow for outgoing message computation (see Section 2.4.4.2).

Observing that the variation in the map is concentrated at the walls of the environment, we choose inducing points to be close to the walls. From each LIDAR scan we extract a set of candidate inducing points by concatenating: i) the input locations for which density is 1 (positive examples), and ii) positive examples perturbed a small distance back towards the robot (negative examples). We choose new inducing points from this shortlist using the greedy variance method. We also randomly select a small number of observations with zero density as inducing points, to sparsely cover free-space. The final accumulated inducing points can be seen in Figure 5.8d.

Upon making contact, two robots connect via a shared factor as per Section 5.3.2. Each robot also subsamples a set of the other's nearby inducing point locations and creates new inducing points in their own model at these locations. Further, as the trajectories are partially overlapping, the robots exchange inducing point posteriors, allowing them to predict in regions observed by the other without having to incorporate all their inducing points into their own model. All robots use a Matérn-1/2 kernel with fixed lengthscale of 5% of the environment length.

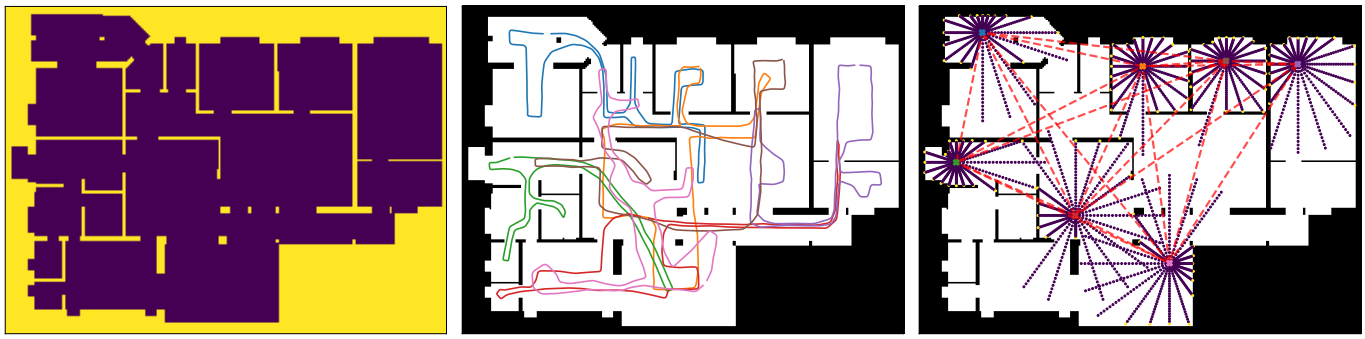
**Evaluation** While our method generates a map distributed between robots, DiNNO produces an estimate of the global map on each robot. We therefore use two modes of evaluation:

1. **Global map accuracy**. As in Yu et al. [2022], we assess DiNNO by the average accuracy of all robots' global maps. For our method, we construct the global map which would be achieved by the robots exchanging both i) their own inducing point posterior and ii) posteriors received from others, when connecting.

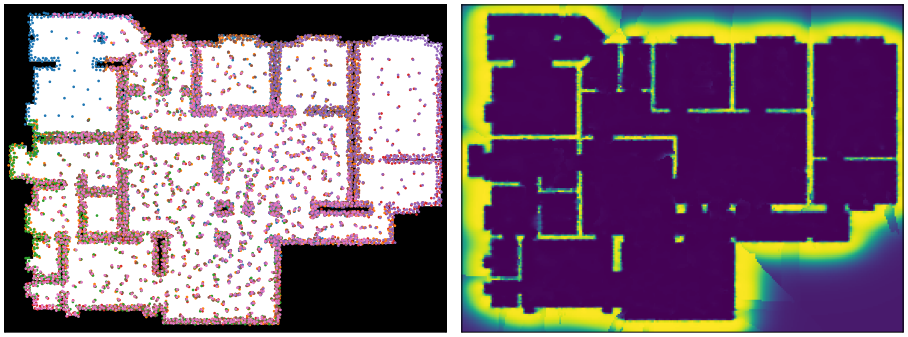
2. **Distributed map accuracy.** In each evaluation round, we asses the accuracy when predicting each test point using the closest robot. As the locations change over time, the same test point may be predicted by different robots over the course of the experiment.

**Results** The maps (Figs 5.8e, f and g) show that our method is able to converge to an accurate map in a single pass, i.e. with each robot traversing their trajectory once. In contrast, DiNNO must repeat the trajectories hundreds of times to get close to the ground truth. We further note that our final map more closely matches the ground truth under both metrics (Figure 5.9), and the distributed and centralised predictions perform almost identically for both methods.

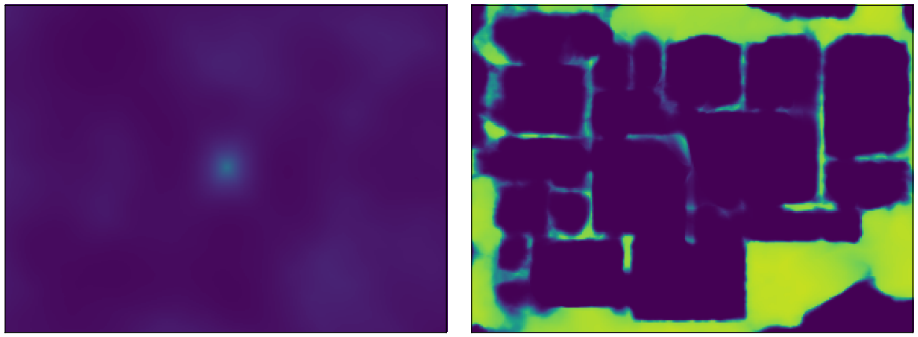
GPs are generally well-suited to learning smooth functions, but are often considered to be poor models of discontinous functions [see e.g Park, 2022]. It may thus be surprising that our distributed GP model performs well on this task, where the ground truth map is clearly discontinous. We attribute this success to two factors. First, we use a Matérn-1/2 kernel, which is better able to capture discontinuity than the commonly used radial basis function kernel. Second, we use a relatively large number of inducing points (see inducing point locations in Figure 5.8d), concentrated around the walls of the environment. This provides sufficient model capacity in the important regions of the map.



(a) Ground truth density (b) Robot trajectories (c) Example LIDAR scans



(d) DistGP inducing points (e) DistGP (ours), single pass



(f) DiNNO, single pass (g) DiNNO, 200 passes

Figure 5.8: **Distributed occupancy mapping.** The robots follow the trajectories in (b), making a LIDAR scan (c) at each timestep to generate the observations from which to learn the map. The red lines (c) mark a typical robot communication pattern. The inducing point locations (d) are colour-coded according to which robot they belong to, consistent with the robot trajectory colours. Our method converges to an accurate map in a single pass through the robots' trajectories (e), as observations are fully fused into the model. In contrast, DiNNO trains NNs iteratively and therefore requires many passes to overcome catastrophic forgetting – (f) and (g). We note that our map at convergence is also more accurate than DiNNO's. Best viewed digitally.

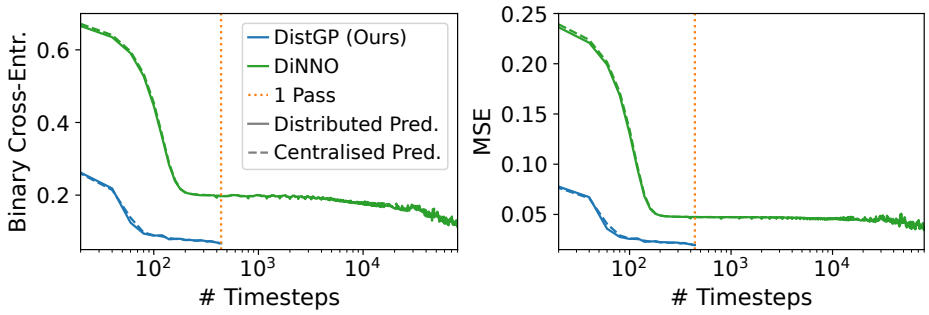



Figure 5.9: **Evolution of occupancy mapping test error**. Note the horizontal axis is log-scale.

## 5.5 Related Work

Many methods build distributed implicit neural maps through inter-robot collaboration [Yu et al., 2022, Deng et al., 2024, Asadi et al., 2024, Zhao et al., 2024, 2025], either using consensus ADMM or direct regularisation to constrain NN weights to be similar. We believe our experiments (Section 5.4) show the benefit of using inducing points with local consistency constraints, rather than global weight-space agreement. Of the distributed NN methods, most similar to ours is DiNNO [Yu et al., 2022] which is a general method for multi-robot function learning rather than a specific method for distributed 3D reconstruction.

In DiNNO, each robot maintains a NN estimate of the global map, which are brought into agreement by consensus ADMM. In contrast, we seek to build a distributed map: each robot maintains their local segment of the map and inter-robot interactions are used to ensure each segment is consistent with its neighbours. We argue our approach is

1. **more scalable.** For problems with large areas and large numbers of robots, each robot having a global map is inefficient and reaching consensus is challenging.

2. **more modular.** We divide the map into segments which can be individually copied and communicated as required. In contrast, it is unclear how a map stored in the weights of a NN can be subdivided.

3. **also able to generate a shared global map.** In the extreme case that each robot shares both its own inducing point posterior, and all the others it has received thus far, we can achieve the same global map sharing as DiNNO.

Other types of base model have been employed for multi-robot learning. As in this work, [Luo and Sycara, 2018] use a distributed GP. Each robot first trains a local GP on its observations and they are then combined as a mixture of GPs, with location-varying mixture weights. The Expectation Maximisation algorithm [Dempster et al., 1977] is used during training to adjust the local models in tandem with the mixture weights of the different observations. Unlike our method, full connectivity is required for normalisation of the mixture weights, and all robots must generate predictions for any new test point. In addition, the use of dense GPs precludes scaling to large datasets.

Recent work has shown GBP to be an effective distributed solver. Our work draws inspiration from Murai et al. [2023], in using local communication between robots to enable collaborative estimation. While Murai et al. [2023] estimate the locations of each robot,

we estimate a distributed map. Patwardhan and Davison [2024] propose a multi-robot mapping method based on GBP. The input space is divided into a regular grid and each robot maintains variables which represent the function value in its cells. These are updated via i) direct observation by a robot measurement or ii) inter-robot communication where two robots add consistency factors between their pairs of grid variables. This method is not able to generalise to unobserved regions and has fixed resolution.

## 5.6 Conclusion

We have presented a new method for multi-robot mapping, in which each robot learns an sparse GP and communicates with others via GBP to ensure consistency. Our method allows for efficient, online learning and uses only local, asynchronous communication. The locality and the probabilistic nature of the inducing points enable efficient learning and communication compared to NN-based methods.

There are many possibilities to build on this work. First, GPs are known to provide accurate uncertainty estimates in many settings, making them effective surrogate models for Bayesian optimisation, and useful for active learning. Our method could be an exciting opportunity for distributed Bayesian optimisation or active learning, in which multiple robots explore different subregions of the input space in parallel. Second, we note that each robot learns kernel hyperparameters in isolation. A message passing procedure to facilitate collaborative estimation in loopy graphs could improve performance considerably.

# Chapter 6

# Conclusions and Future Work

In this thesis, we have presented work on Bayesian learning, showing that a probabilistic perspective affords a means to distribute learning via GBP, either between processors or across multiple robots. Further, we have shown that probabilistic principles provide a new perspective on data augmentation (DA), enabling it to be cleanly incorporated into the training of Bayesian neural networks (BNNs).

In Chapter 3, we investigated the observed co-occurrence between DA and the cold posterior effect (CPE) in Bayesian deep learning (DL). Multiple previous works [Noci et al., 2021, Izmailov et al., 2021] have argued that the observed link between DA and the CPE motivates further study into how exactly we should "properly account for DA in a Bayesian sense" [Noci et al., 2021]. We proposed that DA should be treated as a model invariance, rather than an expanded training set, and we derived multiple lower bounds on the log-likelihood of such an invariant model. We empirically evaluated these bounds for Bayesian DL, and found that the CPE persists. This finding falsified the hypothesis that a commonly used invalid DA likelihood is responsible for the CPE.

In Chapter 4, we presented *GBP Learning*, a framework for training deep networks with GBP inference. Our framework allows similar design freedom to DL, where different types of layer module can be composed into a deep factor graph. While the models contain large numbers of cycles and non-linear transformations for expressivity, we show that we can still train them successfully with loopy GBP, via linearisation of the necessary factors. As training with GBP provides an approximate posterior over parameters, incremental learning comes naturally by using the posterior from the past as the prior for the future.

Experimentally, we showed that GBP Learning can improve over traditional, hand-crafted

factor graph methods. Moreover, we found the technique for continual learning to be effective, both i) for improving video denoising where incremental learning over frames resulted in better reconstruction than per-frame learning and ii) for continual image classification. In the latter case, we found (continual) GBP Learning was able to beat an equivalent NN with a replay buffer containing thousands of examples. Further, we showed that training can be executed layerwise asynchronously, with a random ordering of updates resulting in similar performance as synchronous forward/backward sweeps. Similarly, we showed that a model-parallel simulation in which each layer's messages are updated many times without inter-layer communication, maintained similar performance to scheduled sweeps. Last, we demonstrated the potential of hybrid hand-crafted and learnable models with a toy control experiment.

We shifted focus in Chapter 5. Rather than considering layerwise distributed training, we considered learning algorithms that could be distributed over multiple robots. The method we developed enabled global function approximation from local experience, computation and communication via GBP. Our first step was to assess Tree-structured Gaussian processes (TSGPs), a class of models well-suited to distributed learning. We found, however, that they could be improved significantly simply with the introduction of extra edges which create cycles but improve the consistency of the fitted function between regions. To assess potential for real-world multi-robot mapping problems, we then evaluated this model in simulations with dynamic connectivity and asynchronous GBP message passing. Our results showed that this distributed, asynchronous inference can converge to the same performance as a batch-fitted, centralised implementation. To end the chapter, we experimentally compared our approach to DiNNO [Yu et al., 2022], a DL-based approach to multi-robot learning. We found our method outperformed DiNNO for both sea surface temperature and occupancy mapping tasks.

## 6.1 Future Work

We divide the possible future work into three sections: one relating to each of the Chapters 3, 4 and 5. We discuss both existing research that has built on our work and directions which have not yet been investigated.

### 6.1.1 Probabilistic Data Augmentation and the Cold Posterior Effect

While our experimental results suggested that an invalid DA likelihood was not the cause of the CPE, we believe the discussed likelihood bounds remain a useful method for using DA in a probabilistic context. For example, Immer et al. [2022] use this formulation automatically learn the range of invariances to DA transformations in NNs and van der Ouderaa et al. [2024] use it for inferring symmetries from data.

Follow up work on the CPE has argued that it stems from an inability to specify aleatoric uncertainty with standard categorical likelihoods, which leads to poor modelling of curated datasets with near-zero aleatoric uncertainty [Kapoor et al., 2022, Marek et al., 2024]. This is exacerbated by DA, as averaging over the augmentation distribution $p(\mathbf{x}'|\mathbf{x})$ results in less confident predictions. Instead, Kapoor et al. [2022] and Marek et al. [2024] use Dirichlet likelihood functions, in which aleatoric uncertainty *can* be specified. They show that such a model achieves similar performance to one with a standard categorical likelihood with a cooled posterior. Alternatively, Ng et al. [2024] treat temperature as a model parameter, and propose an automatic method for tuning it.

### 6.1.2 GBP Learning

We believe there are a number of interesting future directions for our GBP Learning work (Chapter 4). Our method uses Gaussian factor graph representations of deep networks, and executes GBP inference in order to learn their parameters. We note that the factor graph is an undirected, joint model over all variables (parameters, activations, inputs, ouputs). As such, we postulate that the same learnt parameters should enable answering different probabilistic queries. For example, the parameters learnt by training a network on the vanilla MNIST dataset, should contain knowledge of both the disciminative mapping $p(y|\mathbf{x})$ and the input density $p(\mathbf{x})$. Thus these parameters should be able to e.g. denoise corrupted MNIST images and classify new ones. This would be hybrid generative-discriminative model akin to that of Grathwohl et al. [2019] but with distributed training and prediction via GBP.

Second, as we highlighted in Chapter 4, our proposed method for designing deep factor graphs, is only one approach to do so. We believe there is a large design space of other possibilities to explore. For example, we only used models with scalar variables, which are unable to capture the rich correlation structure of deep networks. Combining highly correlated variables, such as activation variables within the same layer, into vector-valued nodes with dense covariance would likely result in improved performance. Further, one may

be able to achieve more stable inference in deeper models but stacking multiple NN layers into each inter-activation factor.

Last, we note that our experiments all used relatively shallow networks of up to 5 or so layers. We found that models deeper than this often resulted in worse performance. Detailed diagnosis of the message updates may be helpful for determining why this is the case.

### 6.1.3 Distributed GP Models for Multi-robot Mapping

While we demonstrated our method for multi-robot mapping, we did not consider downstream uses of the estimated, distributed map. One potential use case is for distributed optimisation in which the robots simultaneously learn a map whilst seeking the location of its optimum. This could be useful for, e.g. gas source localisation [Duisterhof et al., 2021]. GP models are known to be effective surrogate models for Bayesian optimisation due to well-calibrated uncertainty enabling effective exploration-exploitation compromise.

Further, we note that our method assumes a robot's ground truth location is known when observations of the map are made. In realistic robot settings this is rarely the case, and extending our model to accommodate noisy input locations would improve the applicability for such settings. This model would enable simultaneous localisation and mapping (SLAM), in which we do inference jointly on both input locations and the map. Our collaborative mapping method could be combined with the Robot Web [Murai et al., 2023] method of collaborative localisation for improved estimation of both.

# Bibliography


Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. http://download.tensorflow.org/paper/whitepaper2015.pdf, 2015.

Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, 630(8016): 493–500, 2024.

Ben Adlam, Jasper Snoek, and Samuel L Smith. Cold posteriors and aleatoric uncertainty. *arXiv preprint arXiv:2008.00029*, 2020.

Pratik Agarwal, Gian Diego Tipaldi, Luciano Spinello, Cyrill Stachniss, and Wolfram Burgard. Robust map optimization using dynamic covariance scaling. In *2013 IEEE international conference on robotics and automation*, pages 62–69. Ieee, 2013.

Sameer Agarwal, Keir Mierle, et al. Ceres solver: Tutorial & reference. *Google Inc*, 2(72):8, 2012.

Laurence Aitchison. A statistical theory of cold posteriors in deep neural networks. *arXiv preprint arXiv:2008.05912*, 2020.

Dario Albani, Joris IJsselmuiden, Ramon Haken, and Vito Trianni. Monitoring and mapping with robot swarms for agricultural applications. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2017.

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems*, 32, 2019.

Nick Alonso, Beren Millidge, Jeffrey Krichmar, and Emre O Neftci. A theoretical framework for inference learning. *Advances in Neural Information Processing Systems*, 35:37335–37348, 2022.

Ignacio Alzugaray, Riku Murai, and Andrew Davison. Pixro: Pixel-distributed rotational odometry with gaussian belief propagation. *arXiv preprint arXiv:2406.09726*, 2024.

DG LiMu Andersen and IW Park. Scaling distributed machine learning with the parameter server. In *Proceedings of the Operating Systems Design and Implementation (OSDI)*, 2014.

Mahboubeh Asadi, Kourosh Zareinia, and Sajad Saeedi. Di-nerf: Distributed nerf for collaborative learning with relative pose refinement. *IEEE Robotics and Automation Letters*, 2024.

Gregory Benton, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. Learning invariances in neural networks. *arXiv preprint arXiv:2010.11882*, 2020.

James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multigrain: a unified image embedding for classes and instances. *arXiv preprint arXiv:1902.05509*, 2019.

Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Proceedings of ICC'93-IEEE International Conference on Communications*, volume 2, pages 1064–1070. IEEE, 1993.

Danny Bickson. *Gaussian belief propagation: Theory and aplication*. PhD thesis, The Hebrew University of Jerusalem, 2008.

Christopher M Bishop. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Christopher L Buckley, Chang Sub Kim, Simon McGregor, and Anil K Seth. The free energy principle for action and perception: A mathematical review. *Journal of Mathematical Psychology*, 81:55–79, 2017.

Thang D Bui and Richard E Turner. Tree-structured gaussian process approximations. *Advances in Neural Information Processing Systems*, 27, 2014.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

David R Burt, Carl Edward Rasmussen, and Mark Van Der Wilk. Convergence of sparse variational inference in gaussian processes regression. *Journal of Machine Learning Research*, 21(131):1–63, 2020.

Cerebras. Cerebras. URL https://www.cerebras.net/.

Shuxiao Chen, Edgar Dobriban, and Jane H. Lee. A group-theoretic framework for data augmentation. *Journal of Machine Learning Research*, 21(245):1–71, 2020. URL http://jmlr.org/papers/v21/20-163.html.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

Dami Choi, Alexandre Passos, Christopher J Shallue, and George E Dahl. Faster neural network training with data echoing. *arXiv preprint arXiv:1907.05550*, 2019.

Robert Cowell. Advanced inference in bayesian networks. In *Learning in graphical models*, pages 27–49. Springer, 1998.

Richard T Cox. Probability, frequency and reasonable expectation. *American journal of physics*, 14(1):1–13, 1946.

Lehel Csató and Manfred Opper. Sparse on-line gaussian processes. *Neural computation*, 14 (3):641–668, 2002.

Vanessa D'Amario, Sanjana Srivastava, Tomotake Sasaki, and Xavier Boix. The data efficiency of deep learning is degraded by unnecessary input dimensions. *Frontiers in Computational Neuroscience*, 16:760085, 2022.

Andreas C Damianou, Michalis K Titsias, and Neil Lawrence. Variational inference for latent variables and uncertain inputs in gaussian processes. 2016.

Francesco D'Angelo and Vincent Fortuin. Repulsive deep ensembles are bayesian. *Advances in Neural Information Processing Systems*, 34:3451–3465, 2021.

Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, and Christopher Re. A kernel theory of modern data augmentation. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1528–1537. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/dao19b.html.

A. J. Davison and J. Ortiz. FutureMapping 2: Gaussian Belief Propagation for Spatial AI. *arXiv preprint arXiv:1910.14139*, 2019.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International conference on machine learning*, pages 7480–7512. PMLR, 2023.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Paul Delestrac, Jonathan Miquel, Debjyoti Bhattacharjee, Diksha Moolchandani, Francky Catthoor, Lionel Torres, and David Novo. Analyzing gpu energy consumption in data movement and storage. In *2024 IEEE 35th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 143–151. IEEE, 2024.

Frank Dellaert, Michael Kaess, et al. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

Yinan Deng, Yujie Tang, Yi Yang, Danwei Wang, and Yufeng Yue. Macim: Multi-agent collaborative implicit mapping. *IEEE Robotics and Automation Letters*, 9(5):4369–4376, 2024.

Robert H Dennard, Fritz H Gaensslen, Hwa-Nien Yu, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of solid-state circuits*, 9(5):256–268, 2003.

David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32, 2019.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

Bardienus P Duisterhof, Shushuai Li, Javier Burgués, Vijay Janapa Reddi, and Guido CHE De Croon. Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9099–9106. IEEE, 2021.

Matthew Dunbabin and Lino Marques. Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics & Automation Magazine*, 19(1):24–39, 2012.

Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: informed scheduling for asynchronous message passing. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 165–173, 2006.

Exolabs. Exolabs blog. 2025. URL https://blog.exolabs.net/.

Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2012.

D Fick. Analog compute-in-memory for ai edge inference. In *2022 International Electron Devices Meeting (IEDM)*, pages 21–8. IEEE, 2022.

Stanislav Fort, Andrew Brock, Razvan Pascanu, Soham De, and Samuel L. Smith. Drawing multiple augmentation samples per image during training efficiently decreases test error. *arXiv preprint arXiv:2105.13343*, 2021.

Vincent Fortuin, Adrià Garriga-Alonso, Mark van der Wilk, and Laurence Aitchison. BNNpriors: A library for Bayesian neural network inference with different prior distributions. *Software Impacts*, page 100079, 2021.

Vincent Fortuin, Adrià Garriga-Alonso, Sebastian W Ober, Florian Wenzel, Gunnar Ratsch, Richard E Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited. In *International Conference on Learning Representations*, 2022.

Adam Foster, Rattana Pukdee, and Tom Rainforth. Improving transformation invariance in contrastive representation learning. *arXiv preprint arXiv:2010.09515*, 2020.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Brendan J Frey, Ralf Koetter, and Nemanja Petrovic. Very loopy belief propagation for unwrapping phase images. *Advances in Neural Information Processing Systems*, 14, 2001.

Karl Friston. A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836, 2005.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

Adrià Garriga-Alonso and Vincent Fortuin. Exact Langevin dynamics with stochastic gradients. *arXiv preprint arXiv:2102.01691*, 2021.

Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Dileep George, Wolfgang Lehrach, Ken Kansky, Miguel Lázaro-Gredilla, Christopher Laan, Bhaskara Marthi, Xinghua Lou, Zhaoshi Meng, Yi Liu, Huayan Wang, et al. A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*, 358(6368):eaag2612, 2017.

Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. Ai and memory wall. *IEEE Micro*, 44(3):33–39, 2024.

Agathe Girard and Roderick Murray-Smith. Learning a gaussian process model with uncertain inputs. *Department of Computing Science, University of Glasgow, Tech. Rep. TR-2003-144*, 2003.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

Matías A. Goldin, Samuele Virgili, and Matthew Chalk. Scalable gaussian process inference of neural responses to natural images. *Proceedings of the National Academy of Sciences*, 120(34):e2301150120, 2023. doi: 10.1073/pnas.2301150120. URL https://www.pnas.org/doi/abs/10.1073/pnas.2301150120.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Graphcore. Graphcore. URL https://www.graphcore.ai/.

Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*, 2019.

Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

Peter Grünwald. The safe bayesian. In *International Conference on Algorithmic Learning Theory*, pages 169–183. Springer, 2012.

Peter Grünwald, Thijs Van Ommen, et al. Inconsistency of bayesian inference for misspecified linear models, and a proposal for repairing it. *Bayesian Analysis*, 12(4):1069–1103, 2017.

John M Hammersley and Peter Clifford. Markov fields on finite graphs and lattices. *Unpublished manuscript*, 46, 1971.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, page 282. Citeseer, 2013.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International conference on machine learning*, pages 1861–1869. PMLR, 2015.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: better training with larger batches. *arXiv preprint arXiv:1901.09335*, 2019.

Sara Hooker. The hardware lottery. *Communications of the ACM*, 64(12):58–65, 2021.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Boyin Huang, Chunying Liu, Viva Banzon, Eric Freeman, Garrett Graham, Bill Hankins, Tom Smith, and Huai-Min Zhang. Improvements of the daily optimum interpolation sea surface temperature (doisst) version 2.1. *Journal of Climate*, 34(8):2923–2939, 2021.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

David Hug, Ignacio Alzugaray, and Margarita Chli. Hyperion–a fast, versatile symbolic gaussian belief propagation framework for continuous-time slam. In *European Conference on Computer Vision*, pages 215–231. Springer, 2024.

Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Mohammad Emtiyaz Khan. Scalable marginal likelihood estimation for model selection in deep learning. *arXiv preprint arXiv:2104.04975*, 2021.

Alexander Immer, Tycho van der Ouderaa, Gunnar Rätsch, Vincent Fortuin, and Mark van der Wilk. Invariance learning in deep neural networks with differentiable laplace approximations. *Advances in Neural Information Processing Systems*, 35:12449–12463, 2022.

Saidul Islam, Hanae Elmekki, Ahmed Elsebai, Jamal Bentahar, Nagat Drawel, Gaith Rjoub, and Witold Pedrycz. A comprehensive survey on applications of transformers for deep learning tasks. *Expert Systems with Applications*, 241:122666, 2024.

Pavel Izmailov, Sharad Vikram, Matthew D. Hoffman, and Andrew Gordon Wilson. What are Bayesian neural network posteriors really like? *arXiv:2104.14421*, 2021.

Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore ipu architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413*, 2019.

Jason K Johnson, Danny Bickson, and Danny Dolev. Fixing convergence of gaussian belief propagation. In *2009 IEEE International Symposium on Information Theory*, pages 1674–1678. IEEE, 2009.

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

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Sanyam Kapoor, Wesley J Maddox, Pavel Izmailov, and Andrew G Wilson. On uncertainty, tempering, and data augmentation in bayesian classification. *Advances in neural information processing systems*, 35:18211–18225, 2022.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *Advances in neural information processing systems*, 32, 2019.

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

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

AN Kolmogorov. *Foundations of the theory of probability*. Courier Dover Publications, 1950.

Imre Risi Kondor. *Group theoretical methods in machine learning*. 2008.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

Miguel Lázaro-Gredilla, Yi Liu, D Scott Phoenix, and Dileep George. Hierarchical compositional feature learning. *arXiv preprint arXiv:1611.02252*, 2016.

Miguel Lázaro-Gredilla, Wolfgang Lehrach, Nishad Gothoskar, Guangyao Zhou, Antoine Dedieu, and Dileep George. Query training: Learning a worse model to infer better marginals in undirected graphical models with hidden variables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8252–8260, 2021.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and Fujie Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Deren Lei, Zichen Sun, Yijun Xiao, and William Yang Wang. Implicit regularization of stochastic gradient descent in natural language processing: Observations and implications. *arXiv preprint arXiv:1811.00659*, 2018.

Ben Leimkuhler and Charles Matthews. The canonical distribution and stochastic differential equations. In *Molecular Dynamics*, pages 211–260. Springer, 2015.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Hongyu Li, Liang Ding, Meng Fang, and Dacheng Tao. Revisiting catastrophic forgetting in large language model tuning. *arXiv preprint arXiv:2406.04836*, 2024.

Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, 2, 2007.

Carlo Lucibello, Fabrizio Pittorino, Gabriele Perugini, and Riccardo Zecchina. Deep learning via message passing algorithms based on belief propagation. *Machine Learning: Science and Technology*, 3(3):035005, jul 2022. IOP Publishing.

Wenhao Luo and Katia Sycara. Adaptive sampling and online learning in multi-robot sensor coverage with mixture of gaussian processes. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6359–6364. IEEE, 2018.

Clare Lyle, Mark van der Wilk, Marta Kwiatkowska, Yarin Gal, and Benjamin Bloem-Reddy. On the benefits of invariance in neural networks. *arXiv preprint arXiv:2005.00178*, 2020.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *International Conference on Machine Learning*, pages 23190–23211. PMLR, 2023.

David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

David JC MacKay. Good error-correcting codes based on very sparse matrices. *IEEE transactions on Information Theory*, 45(2):399–431, 2002.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Dan MacKinlay, Russell Tsuchida, Daniel Edward Pagendam, and Petra Kuhnert. Gaussian ensemble belief propagation for efficient inference in high-dimensional, black-box systems. In *The Thirteenth International Conference on Learning Representations*, 2025.

Dmitry M. Malioutov, Jason K. Johnson, and Alan S. Willsky. Walk-sums and belief propagation in gaussian graphical models. *Journal of Machine Learning Research*, 7(73): 2031–2064, 2006. URL http://jmlr.org/papers/v7/malioutov06a.html.

Martin Marek, Brooks Paige, and Pavel Izmailov. Can a confident prior replace a cold posterior? *arXiv preprint arXiv:2403.01272*, 2024.

Martin Marek, Sanae Lotfi, Aditya Somasundaram, Andrew Gordon Wilson, and Micah Goldblum. Small batch size training for language models: When vanilla sgd works, and why gradient accumulation is wasteful. *arXiv preprint arXiv:2507.07101*, 2025.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Robert J. McEliece, David J. C. MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of pearl's" belief propagation" algorithm. *IEEE Journal on selected areas in communications*, 16(2):140–152, 1998.

Andrew McHutchon and Carl Rasmussen. Gaussian process training with input noise. *Advances in Neural Information Processing Systems*, 24:1341–1349, 2011.

Peter L McMahon. The physics of optical computing. *Nature Reviews Physics*, 5(12): 717–734, 2023.

Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. Helix: Serving large language models over heterogeneous gpus and network via max-flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 586–602, 2025.

Beren Millidge, Alexander Tschantz, and Christopher L Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *Neural Computation*, 34(6): 1329–1368, 2022.

Thomas P Minka. Expectation propagation for approximate bayesian inference. *arXiv preprint arXiv:1301.2294*, 2013.

Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 479(480):104, 1969.

Kostadin Mishev, Ana Gjorgjevikj, Irena Vodenska, Lubomir T Chitkushev, and Dimitar Trajanov. Evaluation of sentiment analysis in finance: from lexicons to transformers. *IEEE access*, 8:131662–131682, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990.

Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

Riku Murai, Joseph Ortiz, Sajad Saeedi, Paul HJ Kelly, and Andrew J Davison. A robot web for distributed many-device localization. *IEEE Transactions on Robotics*, 40:121–138, 2023.

Riku Murai, Ignacio Alzugaray, Paul HJ Kelly, and Andrew J Davison. Distributed simultaneous localisation and auto-calibration using gaussian belief propagation. *IEEE Robotics and Automation Letters*, 9(3):2136–2143, 2024.

Kevin Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. *arXiv preprint arXiv:1301.6725*, 2013.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: an empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475, 1999.

Seth Nabarro, Stoil Ganev, Adrià Garriga-Alonso, Vincent Fortuin, Mark van der Wilk, and Laurence Aitchison. Data augmentation in Bayesian neural networks and the cold posterior effect. In *Uncertainty in Artificial Intelligence*, pages 1434–1444. PMLR, 2022.

Seth Nabarro, Mark van der Wilk, and Andrew J Davison. Learning in deep factor graphs with Gaussian belief propagation. In *International Conference on Machine Learning*, pages 37141–37163. PMLR, 2024.

Seth Nabarro, Mark van der Wilk, and Andrew J Davison. A distributed Gaussian process model for multi-robot mapping. Under review at *2026 International Conference on Robotics and Automation (ICRA)*, 2025.

Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 341–346. IEEE, 2016.

Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

Jun Nagata and Yusuke Sekikawa. Tangentially elongated gaussian belief propagation for event-based incremental optical flow estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21940–21949, 2023.

Toshiyuki Nakagaki, Hiroyasu Yamada, and Ágota Tóth. Maze-solving by an amoeboid organism. *Nature*, 407(6803):470–470, 2000.

Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 1995.

Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

Jerzy Neyman. Frequentist probability and frequentist statistics. *Synthese*, 36(1):97–131, 1977.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018.

Kenyon Ng, Chris van der Heide, Liam Hodgkinson, and Susan Wei. Temperature optimization for bayesian deep learning. *arXiv preprint arXiv:2410.05757*, 2024.

Nobel Committee for Chemistry. Computational protein design and protein structure prediction. https://www.nobelprize.org/uploads/2024/10/advanced-chemistryprize2024.pdf, 2024.

Nobel Committee for Physics. For foundational discoveries and inventions that enable machine learning with artificial neural networks. https://www.nobelprize.org/uploads/2024/11/advanced-physicsprize2024-3.pdf, 2024.

Lorenzo Noci, Kevin Roth, Gregor Bachmann, Sebastian Nowozin, and Thomas Hofmann. Disentangling the roles of curation, data-augmentation and the prior in the cold posterior effect. *arXiv preprint arXiv:2106.06596*, 2021.

Sebastian W Ober and Laurence Aitchison. Global inducing point variational posteriors for Bayesian neural networks and deep Gaussian processes. *arXiv preprint arXiv:2005.08140*, 2020.

A ŌHagan. On curve fitting and optimal design for regression. *J. Royal Stat. Soc. B*, 40: 1–32, 1978.

Joseph Ortiz, Mark Pupilli, Stefan Leutenegger, and Andrew J Davison. Bundle adjustment on a graph processor. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2416–2425, 2020.

Joseph Ortiz, Talfan Evans, and Andrew J Davison. A visual introduction to gaussian belief propagation. *arXiv preprint arXiv:2107.02308*, 2021.

Joseph Ortiz, Talfan Evans, Edgar Sucar, and Andrew J Davison. Incremental abstraction in distributed probabilistic slam graphs. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7566–7572. IEEE, 2022.

Kazuki Osawa, Siddharth Swaroop, Anirudh Jain, Runa Eschenhagen, Richard E Turner, Rio Yokota, and Mohammad Emtiyaz Khan. Practical deep learning with Bayesian principles. *arXiv preprint arXiv:1906.02506*, 2019.

Chiwoo Park. Jump gaussian process model for estimating piecewise continuous regression functions. *Journal of Machine Learning Research*, 23(278):1–37, 2022.

Thomas Parr, Dimitrije Markovic, Stefan J Kiebel, and Karl J Friston. Neuronal message passing using mean-field, bethe, and marginal approximations. *Scientific reports*, 9(1): 1889, 2019.

Aalok Patwardhan and Andrew J Davison. A distributed multi-robot framework for exploration, information acquisition and consensus. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12062–12068. IEEE, 2024.

Aalok Patwardhan, Riku Murai, and Andrew J Davison. Distributing collaborative multi-robot planning with gaussian belief propagation. *IEEE Robotics and Automation Letters*, 8(2):552–559, 2022.

Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the AAAI National Conference on AI*, pages 133–136, 1982.

Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 724–732, 2016.

Joshua C Peterson, Ruairidh M Battleday, Thomas L Griffiths, and Olga Russakovsky. Human uncertainty makes classification more robust. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9617–9626, 2019.

Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Tom R Andersson, Andrew El-Kadi, Dominic Masters, Timo Ewalds, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, et al. Probabilistic weather forecasting with machine learning. *Nature*, 637(8044):84–90, 2025.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Ananth Ranganathan, Michael Kaess, and Frank Dellaert. Loopy sam. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2191–2196, 2007.

Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1): 79–87, 1999.

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

Richard W Reynolds, Thomas M Smith, Chunying Liu, Dudley B Chelton, Kenneth S Casey, and Michael G Schlax. Daily high-resolution-blended analyses for sea surface temperature. *Journal of climate*, 20(22):5473–5496, 2007.

Callum Rhodes, Cunjia Liu, and Wen-Hua Chen. Structurally aware 3d gas distribution mapping using belief propagation: A real-time algorithm for robotic deployment. *IEEE Transactions on Automation Science and Engineering*, 21(2):1623–1637, 2023.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th international conference on learning representations, ICLR 2018-conference track proceedings*, volume 6. International Conference on Representation Learning, 2018.

Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15):e2016239118, 2021.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Tommaso Salvatori, Ankur Mali, Christopher L Buckley, Thomas Lukasiewicz, Rajesh PN Rao, Karl Friston, and Alexander Ororbia. Brain-inspired computational intelligence via predictive coding. *arXiv preprint arXiv:2308.07870*, 2023.

Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 685–693. PMLR, 2021.

Leonard J Savage. *The foundations of statistics*. Courier Corporation, 1954.

Sambu Seo, M. Wallat, T. Graepel, and K. Obermayer. Gaussian process regression: active data selection and test point rejection. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 241–246 vol.3, 2000.

John Shawe-Taylor. Symmetries and discriminability in feedforward network architectures. *IEEE Transactions on Neural Networks*, 4(5):816–826, 1993.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18, 2005.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

H. Sutter. Welcome to the jungle. URL https://herbsutter.com/welcome-to-the-jungle, 2011.

Charles Sutton and Andrew McCallum. Improved dynamic schedules for belief propagation. *arXiv preprint arXiv:1206.5291*, 2012.

Richard Sutton. The bitter lesson. *Incomplete Ideas (blog)*, 13(1):38, 2019.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Naif Tarafdar, Giuseppe Di Guglielmo, Philip C Harris, Jeffrey D Krupa, Vladimir Loncar, Dylan S Rankin, Nhan Tran, Zhenbin Wu, Qianfeng Shen, and Paul Chow. Aigean: An open framework for machine learning on heterogeneous clusters. In *FCCM conference proceedings*, 2020.

Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pages 402–419. Springer, 2020.

Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021.

J. Tukey. A survey of sampling from contaminated distributions. *Contributions to Probability and Statistics.*, 1960.

Ali Unlu and Laurence Aitchison. Gradient regularization as approximate variational inference. *Entropy*, 23(12):1629, 2021.

Gido M van de Ven, Nicholas Soures, and Dhireesha Kudithipudi. Continual learning and catastrophic forgetting. *arXiv preprint arXiv:2403.05175*, 2024.

Tycho F van der Ouderaa, Mark van der Wilk, and Pim De Haan. Noether's razor: Learning conserved quantities. *Advances in Neural Information Processing Systems*, 37: 135943–135965, 2024.

Mark van der Wilk, Matthias Bauer, ST John, and James Hensman. Learning invariances using the marginal likelihood. *arXiv preprint arXiv:1808.05563*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation by pseudo-moment matching. In *International Workshop on Artificial Intelligence and Statistics*, pages 308–315. PMLR, 2003a.

Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on information theory*, 49(5):1120–1146, 2003b.

Peng Wang, Xiaohui Shen, Zhe Lin, Scott Cohen, Brian Price, and Alan L Yuille. Towards unified depth and semantic prediction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2800–2809, 2015.

Xi Wang and Laurence Aitchison. Bayesian ood detection with aleatoric uncertainty and outlier exposure. 2021.

Yige Wang, Juntan Zhang, Marc Fossorier, and Jonathan S Yedidia. Reduced latency iterative decoding of ldpc codes. In *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, volume 3, pages 6–pp. IEEE, 2005.

Y Weiss and WT Freeman. Correctness of belief propagation in gaussian models of arbitrary topology. Technical report, Technical Report TR UCB//CSD-99-1046, University of California at Berkeley, 1999.

Johannes Welbl. Casting random forests as artificial neural networks (and profiting from it). In *German Conference on Pattern Recognition*, pages 765–771. Springer, 2014.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Max Welling and Yee Whye Teh. Belief optimization for binary networks: A stable alternative to loopy belief propagation. In *UAI*, pages 554–561, 2001.

Max Welling, Michal Rosen-zvi, and Geoffrey E Hinton. Exponential family harmoniums with an application to information retrieval. In L. Saul, Y. weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004. URL https://proceedings.neurips.cc/paper/2004/file/0e900ad84f63618452210ab8baae0218-Paper.pdf.

Florian Wenzel, Kevin Roth, Bastiaan S Veeling, Jakub Świątkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the Bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*, 2020.

Paul J Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

Max A Woodbury. *Inverting modified matrices*. Department of Statistics, Princeton University, 1950.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review*, 50:100595, 2023.

Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on information theory*, 51(7):2282–2312, 2005.

Logan Yliniemi, Adrian K Agogino, and Kagan Tumer. Multirobot coordination for space exploration. *AI Magazine*, 35(4):61–74, 2014.

KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 868–875. IEEE, 2019.

Javier Yu, Joseph A Vincent, and Mac Schwager. Dinno: Distributed neural network optimization for multi-robot collaborative learning. *IEEE Robotics and Automation Letters*, 7(2):1896–1903, 2022.

Yuexiang Zhai, Shengbang Tong, Xiao Li, Mu Cai, Qing Qu, Yong Jae Lee, and Yi Ma. Investigating the catastrophic forgetting in multimodal large language model fine-tuning. In *Conference on Parsimony and Learning*, pages 202–227. PMLR, 2024.

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, pages 5852–5861. PMLR, 2018.

Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient MCMC for Bayesian deep learning. *arXiv preprint arXiv:1902.03932*, 2019.

Hongrui Zhao, Boris Ivanovic, and Negar Mehr. Distributed nerf learning for collaborative multi-robot perception. *arXiv preprint arXiv:2409.20289*, 2024.

Hongrui Zhao, Boris Ivanovic, and Negar Mehr. Ramen: Real-time asynchronous multi-agent neural implicit mapping. *arXiv preprint arXiv:2502.19592*, 2025.

# Appendix A

# Appendix for Data augmentation in Bayesian neural networks and the cold posterior effect

## A.1 Data Augmentation Log-likelihood Bounds for Non-Bayesian Neural Networks

This appendix contains the experimental evaluation of the data augmentation lower bounds proposed in Chapter 3 for non-Bayesian neural networks. The work was carried out by Stoil Ganev, and forms part of our paper *Data augmentation in Bayesian neural networks and the cold posterior effect* [Nabarro et al., 2022]. It is included for completeness here.

We restate the log-likelihood bounds from Chapter 3:

$$\hat{L}^{i}_{\text{prob},K}\left(y_i;\theta\right) = \log\left(\frac{1}{K}\sum_{k=1}^{K}\text{softmax}_{y_i}\,\mathbf{f}(\mathbf{x}'_{i;k};\theta)\right), \tag{A.1}$$

$$\hat{L}^{i}_{\text{logits},K}\left(y_i;\theta\right) = \log\text{softmax}_{y_i}\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{f}(\mathbf{x}'_{i;k};\theta)\right), \tag{A.2}$$

$$\mathbf{x}'_{i;k} \sim p\left(\mathbf{x}'_i\middle|\mathbf{x}_i\right), k = 1,\ldots,K \tag{A.3}$$

where $p\left(\mathbf{x}'_i|\mathbf{x}_i\right)$ is the distribution over augmentations of $\mathbf{x}_i$.

We compare averaging logits and averaging probabilities in a non-Bayesian setting: SGD

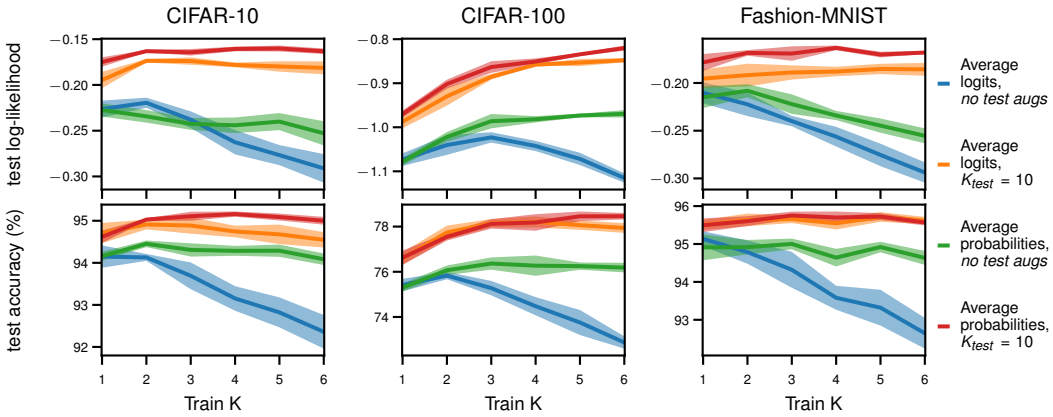



Figure A.1: **Image classification results for the DA likelihood bounds with SGD training**. We compare averaging logits and probabilities for different values of $K_{\text{train}}$, and using $K_{\text{test}} = 10$ vs. using no test-time augmentations, for a ResNet18 with SGD (i.e. no Bayesian inference). We use only full orbit DA to decouple $K_{\text{train}}$ from $K_{\text{test}}$.

training. Higher values of $K_{\text{train}}$ imply a larger computational cost per epoch, as each image is replicated and augmented $K_{\text{train}}$ times before going through the network. When assessing the benefit of averaging probabilities/logits over standard DA for SGD training, we must therefore control for computational cost. We do this by training for $200/K_{\text{train}}$ epochs. Note that $K_{\text{train}} = 1$ with no test-time augmentation (i.e. green and blue in Fig. A.1) corresponds to the standard DA approach for both averaging logits and averaging probabilities (3.21). In this experiment, we consider only the full orbit setting, which unlike finite orbit allows us to decouple $K_{\text{train}}$ and $K_{\text{test}}$.

We trained ResNet18 on CIFAR-10, CIFAR-100 [Krizhevsky et al., 2009] and FashionMNIST [Xiao et al., 2017] with a learning rate of 0.1, decayed to 0.01 three quarters of the way through training. We apply the same two augmentation transformations as Wenzel et al. [2020], Fortuin et al. [2022], Noci et al. [2021]: 1. a random crop with padding of four pixels on all borders and 2. a random horizontal flip with probability 0.5. The training runs took around 12 GPU-days on Nvidia 2080s.

In agreement with past work [Lyle et al., 2020], we found that averaging over augmentations at test-time (red and orange) is better than using the test image without augmentation (green and blue), with $K_{\text{train}} = 1$ corresponding to the standard DA procedure. In addition, we show that improved performance with multiple test-time augmentations continues to hold for larger values of $K_{\text{train}}$. Thus, if sufficient compute is available at test-time, averaging

across augmentations gives an easy method to improve the performance of a pre-trained network.

Importantly, we see some performance gains with higher values of $K_{\text{train}}$ if we focus on the case with test augmentations, though they are somewhat inconsistent across datasets. We see strong improvements for the hardest dataset (CIFAR-100), and smaller improvements that saturate at $K_{\text{train}} = 2$ for CIFAR-10. For FashionMNIST, the picture is more mixed. We suspect this is because we used a DA strategy tuned for CIFAR-10 and CIFAR-100, rather than FashionMNIST.

In addition, averaging probabilities seems to give somewhat better performance than averaging logits: compare averaging probabilities vs. logits both with test-time augmentation (red vs. orange) and without test-time augmentation (green vs. blue). The performance differences are consistent in both comparisons, though smaller when test-time augmentation is applied.

Indeed, performance falls quite dramatically as $K_{\text{train}}$ increases for averaging logits without test-time augmentation (blue). This is an indication that averaging probabilities and logits might actually behave quite differently. To understand how these differences might arise, consider the effect of averaging on the NN function itself. Both schemes can be justified by using averaging to increase invariance to the augmentation transformations (Sec. 3.2.2). Averaging probabilities, however, also forces the NN function itself to become invariant. If different augmentations produce different predictions, then the resulting averaged class probabilities will be more uncertain, which is penalized by the likelihood on the training points. This effect is much weaker when averaging logits. Consider an extreme example, as illustrated in Fig. A.2. It is a two-class classification problem with two augmentations, $\mathbf{x}'_1$ and $\mathbf{x}'_2$, of the same image with logits, $\mathbf{f}(\mathbf{x}'_1) = (10, -10)$ and $\mathbf{f}(\mathbf{x}'_2) = (-1, 1)$. Averaging logits gives us $\mathbb{E}\left[\mathbf{f}(\mathbf{x}')\right] = (4.5, -4.5)$, and applying the softmax, we very confidently predict the first class. In contrast, if we use averaging probabilities, then the first augmentation almost certainly predicts the first class $p(\mathbf{x}'_1) \approx (1, 0)$ and the second augmentation almost certainly predicts the second class, $p(\mathbf{x}'_2) \approx (0, 1)$, so when we average them we obtain $\mathbb{E}\left[p(\mathbf{x}')\right] \approx (0.5, 0.5)$, which indicates a high degree of uncertainty.

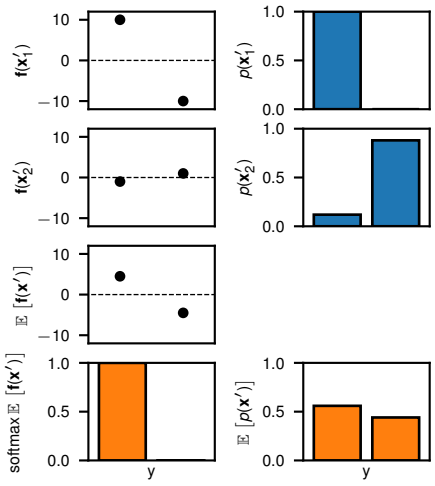
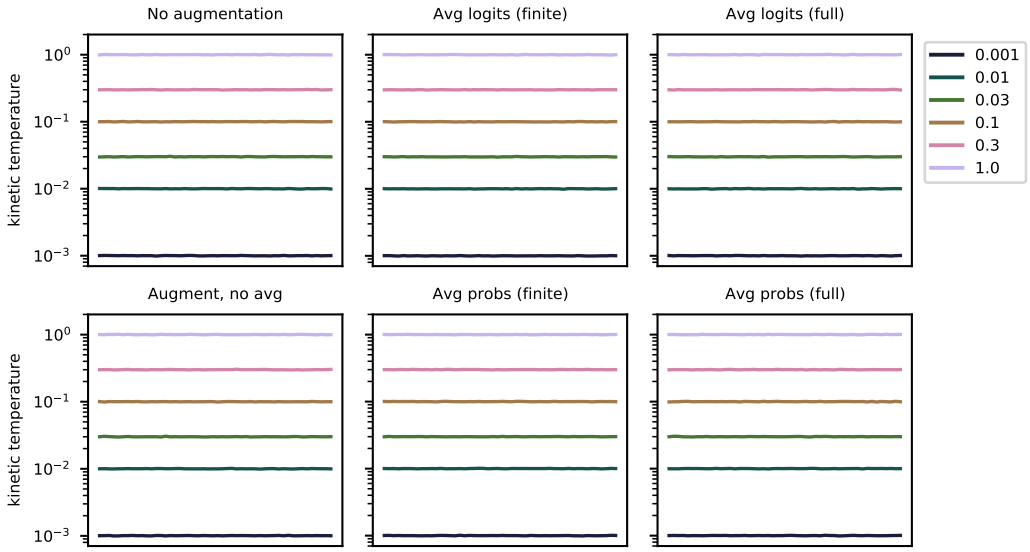
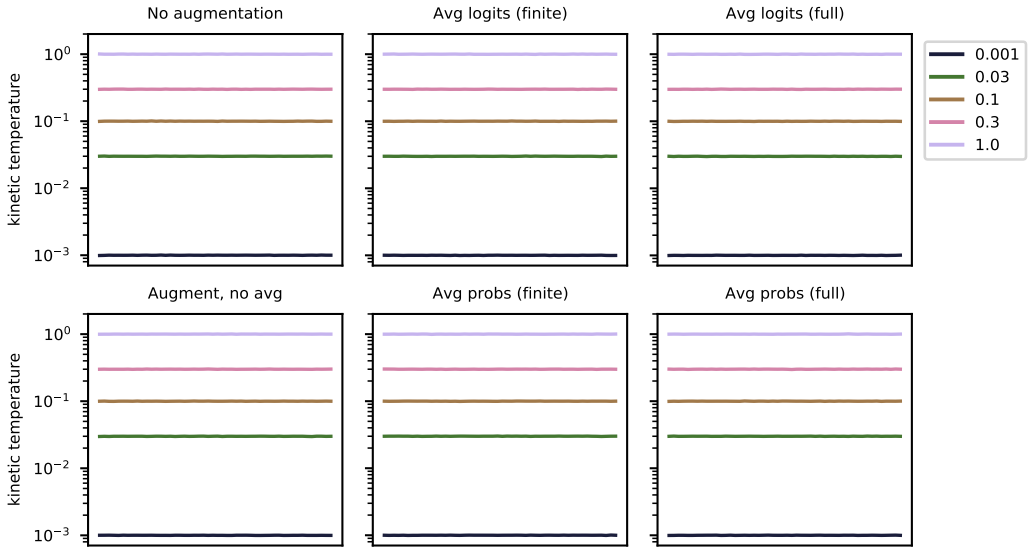



Figure A.2: **Example illustrating the difference between averaging logits and averaging probabilities**. $\mathbf{x}'_1$ and $\mathbf{x}'_2$ are two augmentations of the same image, $\mathbf{f}(\mathbf{x}'_1)$ and $\mathbf{f}(\mathbf{x}'_2)$ are logits outputted by a NN, and $p(\mathbf{x}'_1)$ and $p(\mathbf{x}'_2)$ are the probabilities corresponding to these logits. The prediction derived from the averaged logits is much more certain than the average of the individual probabilities.

## A.2 Kinetic Temperature Diagnostic Results




(a) MNIST, FCNN.




(b) CIFAR-10, ResNet20.

Figure A.3: **The evolution of the kinetic temperature diagnostic [Leimkuhler and Matthews, 2015] during sampling** (horizontal axis represents MCMC step). Good agreement between the diagnostic temperature and intended temperature (in legend) suggests accurate inference. Plot generated by Adriá Garriga-Alonso.

# Appendix B

# Appendix for Learning in Deep Factor Graphs with Gaussian Belief Propagation

## B.1 Factor-to-variable Message Update Optimisation

We aim to reduce the complexity of message update computations (2.66), (2.67) for the factors we describe in Sections 4.2.1. We note these factors generally have low dimensional observations $\mathbf{y}_k$ but are connected to many variables.

The bottleneck of the factor-to-variable message is the inversion of the summed matrices: $\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)} = \Lambda_k^{(\setminus i,\setminus i)} + \Lambda_{m_{\setminus i}}$, i.e. the sum of the partial factor precision matrix $\Lambda_k^{(\setminus i,\setminus i)}$ and the diagonal matrix of incoming messages from all variables except recipient $i$, matrix $\Lambda_{m_{\setminus i}}$. Naïve inversion of this matrix has complexity $\mathcal{O}\left(|\text{nei}\left(\phi_k\right)|^3\right)$.

We note this can be reduced to $\mathcal{O}\left(|\text{nei}\left(\phi_k\right)|M_k^2 + M_k^3\right)$, where $M_k$ is the dimensionality of the observation vector ($M_k := \dim \mathbf{y}_k$). This speedup is achieved by substituting $\Lambda_k$ with the linearised factor precision (2.77) and applying the Woodbury identity [Woodbury, 1950]:

$$\Lambda_{k+m} = J_k^\intercal \Lambda_{\mathbf{y}_k} J_k + \Lambda_m \tag{B.1}$$

$$\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)} = (J_k^{(:,\setminus i)})^\intercal \Lambda_{\mathbf{y}_k} J_k^{(:,\setminus i)} + \Lambda_{m_{\setminus i}} \tag{B.2}$$

$$\left(\Lambda_{k+m_{\setminus i}}^{(\setminus i,\setminus i)}\right)^{-1} = \Lambda_{m_{\setminus i}}^{-1} - \Lambda_{m_{\setminus i}}^{-1}(J_k^{(:,\setminus i)})^\intercal \left(\Lambda_{\mathbf{y}_k}^{-1} + J_k^{(:,\setminus i)}\Lambda_{m_{\setminus i}}^{-1}(J_k^{(:,\setminus i)})^\intercal\right)^{-1} J_k^{(:,\setminus i)}\Lambda_{m_{\setminus i}}^{-1}\,, \tag{B.3}$$

where $J_k^{(:,\setminus i)}$ denotes all but the $i$th column of the factor Jacobian. This expression only features the inversion of an $M_k \times M_k$ matrix, rather than a $|\text{nei}(\phi_k)| \times |\text{nei}(\phi_k)|$ matrix.

Further efficiencies come from substituting (B.3) back into the factor-to-variable message updates (2.66), (2.67)

$$\eta_{\phi_k \to x_i} = \eta_k^{(i)} - \Lambda_{k+m_{\setminus i}}^{(i,\setminus i)} \left( \Lambda_{m_{\setminus i}}^{-1} - \Lambda_{m_{\setminus i}}^{-1} (J_k^{(:,\setminus i)})^\intercal \left( \Lambda_{\mathbf{y}_k}^{-1} + J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} (J_k^{(:,\setminus i)})^\intercal \right)^{-1} J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} \right) \eta_{k+m_{\setminus i}}^{(\setminus i)} \tag{B.4}$$

$$\Lambda_{\phi_k \to x_i} = \Lambda_k^{(i,i)} - \Lambda_{k+m_{\setminus i}}^{(i,\setminus i)} \left( \Lambda_{m_{\setminus i}}^{-1} - \Lambda_{m_{\setminus i}}^{-1} (J_k^{(:,\setminus i)})^\intercal \left( \Lambda_{\mathbf{y}_k}^{-1} + J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} (J_k^{(:,\setminus i)})^\intercal \right)^{-1} J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} \right) \Lambda_{k+m_{\setminus i}}^{(\setminus i,i)} . \tag{B.5}$$

Noting that $\Lambda_{k+m_{\setminus i}}^{(i,\setminus i)}$ contains only off-diagonal elements and so only contains elements of $\Lambda_k$ (and not $\Lambda_{m_{\setminus i}}$), we can then write $\Lambda_{k+m_{\setminus i}}^{(i,\setminus i)} = (J_k^{(:,i)})^\intercal \Lambda_{\mathbf{y}_k} J_k^{(:,\setminus i)}$. Substituting this into the above expression gives

$$\eta_{\phi_k \to x_i} = \eta_k^{(i)} - (J_k^{(:,i)})^\intercal \Lambda_{\mathbf{y}_k} J_k^{(:,\setminus i)} \left( \Lambda_{m_{\setminus i}}^{-1} - \Lambda_{m_{\setminus i}}^{-1} (J_k^{(:,\setminus i)})^\intercal \left( \Lambda_{\mathbf{y}_k}^{-1} + J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} (J_k^{(:,\setminus i)})^\intercal \right)^{-1} J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} \right) \eta_{k+m_{\setminus i}}^{(\setminus i)} \tag{B.6}$$

$$\Lambda_{\phi_k \to x_i} = \Lambda_k^{(i,i)} - \tag{B.7}$$

$$(J_k^{(:,i)})^\intercal \Lambda_{\mathbf{y}_k} J_k^{(:,\setminus i)} \left( \Lambda_{m_{\setminus i}}^{-1} - \Lambda_{m_{\setminus i}}^{-1} (J_k^{(:,\setminus i)})^\intercal \left( \Lambda_{\mathbf{y}_k}^{-1} + J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} (J_k^{(:,\setminus i)})^\intercal \right)^{-1} J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} \right) (J_k^{(:,\setminus i)})^\intercal \Lambda_{\mathbf{y}_k} J_k^{(:,i)} . \tag{B.8}$$

Right-multiplying the $J_k^{(:,\setminus i)}$ before the parentheses, and left multiplying the $(J_k^{(:,\setminus i)})^\intercal$ after gives

$$\eta_{\phi_k \to x_i} = \eta_k^{(i)} - \left( J_k^{(:,i)} \right)^\intercal \Lambda_{\mathbf{y}_k} \left( T_i - U_i \left( \Lambda_{\mathbf{y}_k}^{-1} - U_i \right)^{-1} T_i \right) \tag{B.9}$$

$$\Lambda_{\phi_k \to x_i} = \Lambda_k^{(i,i)} - \left( J_k^{(:,i)} \right)^\intercal \Lambda_{\mathbf{y}_k} \left( U_i - U_i \left( \Lambda_{\mathbf{y}_k}^{-1} - U_i \right)^{-1} U_i \right) \Lambda_{\mathbf{y}_k} J_k^{(:,i)} , \tag{B.10}$$

where we have defined $U_i := J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} \left( J_k^{(:,\setminus i)} \right)^\intercal$ and $T_i := J_k^{(:,\setminus i)} \Lambda_{m_{\setminus i}}^{-1} \eta_{k+m_{\setminus i}}^{(\setminus i)}$.

Given $U_i$ and $T_i$, both message updates are $\mathcal{O}\left(M_k^3\right)$, i.e. independent of $|\text{nei}(\phi_k)|$. However direct computation of $U_i$ or $T_i$ has complexity $\mathcal{O}\left(|\text{nei}(\phi_k)| M_k^2\right)$ for each outgoing variable $i$ ($\Lambda_m$ is diagonal), so the overall complexity is quadratic in $|\text{nei}(\phi_k)|$. We achieve

linear complexity in $|\text{nei}(\phi_k)|$ by exploiting the relation $U_i = J_k \Lambda_m^{-1} J_k^\intercal - J_k^{(:,i)} \Lambda_{m_i}^{-1} (J_k^{(:,i)})^\intercal$ where $J_k \Lambda_m^{-1} J_k^\intercal$ can be computed once for all connected variables. Similarly, we use $T_i = J_k \Lambda_m^{-1} \eta_{k+m_{\setminus i}} - J_k^{(:,i)} \Lambda_{m_i}^{-1} \eta_{k+m_{\setminus i}}^{(i)}$ for the information update. This reduces the complexity for updating all outgoing messages from a factor from $\mathcal{O}\left(|\text{nei}(\phi_k)| \cdot \left(|\text{nei}(\phi_k)| \cdot M_k^2 + M_k^3\right)\right)$ to $\mathcal{O}\left(|\text{nei}(\phi_k)| M_k^3\right)$. In addition, these optimisations require memory $\mathcal{O}\left(|\text{nei}(\phi_k)| \cdot M_k + M_k^2\right)$ which, in most cases, is a saving relative to $\mathcal{O}\left(|\text{nei}(\phi_k)|^2\right)$ needed to store the full factor precision.