The Dissertation Committee for Evan Austin Ott
certifies that this is the approved version of the following dissertation:

Probabilistic Neural Networks

Committee:

Sinead Williamson, Supervisor

José Miguel Hernández-Lobato

James Scott

Mingyaun Zhou

Probabilistic Neural Networks

by

Evan Austin Ott

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2022

Dedication

For Melody. Thank you for being by my side this whole time.

Acknowledgments

I acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing High Performance Computing resources that have contributed to the research results reported within this dissertation.

I completed portions of this work while supported by the National Library of Medicine of the National Institutes of Health under Award Number `5T32LM012414-02`. The content is solely the responsibility of the author and does not necessarily represent the official views of the National Institutes of Health. More information about this funding award is available at `reporter.nih.gov/project-details/9248416`.

I also want to say a brief word of thanks to everyone who supported me on this journey. Thank you to my parents, my family, my friends, and my teachers. Also a special thanks to Sinead for her support in advising me throughout my time in the doctoral program.

Abstract

Probabilistic Neural Networks

Evan Austin Ott, Ph.D.
The University of Texas at Austin, 2022

Supervisor: Sinead Williamson

Neural networks are flexible models capable of capturing complicated data relationships. However, neural networks are typically trained to make maximum likelihood predictions that ignore uncertainty in the model parameters. Additionally, stochasticity is often not incorporated into predictions. Inspired by Bayesian methodology, this work explores ways of incorporating uncertainty in neural network-based models, whether approximating a Bayesian posterior, formulating an alternative to a Bayesian posterior, or developing generative models inspired by parametric Bayesian models.

First, we explore the impact of different approximations in approximate Bayesian inference by considering probabilistic backpropagation (Hernández-Lobato and Adams, 2015), an approximate method for Bayesian neural networks that uses several Gaussian approximations in an assumed density filtering (Opper, 1999) setting. We explore an alternative approximation using a spike-and-slab distribution, designed to be a more accurate approximation to the true distribution.

Second, we explore the use of alternative notions of a posterior distribution. Nonparametric learning (Lyddon et al., 2019; Fong et al., 2019) is a method that provides principled uncertainty estimates about parameters of interest, while making minimal assumptions about the parameters. However, a naïve approach will scale

poorly to large models such as normalizing flows. We show that an approximate implementation, where some parameters are fixed across all samples from the posterior, allows us to achieve improved predictive performance without incurring excessive computational costs.

Finally, building on results from edge-exchangeable graphs (Crane and Dempsey, 2016; Cai et al., 2016) and generative graph neural networks (e.g., Li et al. (2018b)), we propose an edge-based generative graph neural network model. By construction, our model should be able to more easily learn and replicate the structure of sparse graphs, which are common in real-world settings. In this ongoing work, we find that the resulting distributions over graphs are able to capture realistic graph properties in a variety of settings.

Table of Contents

List of Tables

List of Figures

Chapter 1: Summary of Contributions

Chapter 2 provides an extended background for each of the three main research areas that follow. This includes a more in-depth discussion of related work that is important for understanding the breadth and depth of work in the research areas presented here.

Chapter 3 discusses work to extend probabilistic backpropagation Hernández-Lobato and Adams (2015), an approximate method for Bayesian inference in neural networks. The main focus is our attempt to replace a Gaussian approximation with a spike-and-slab to better model sparsity in the network. We find that, while spike-and-slab distributions are a better approximation when working with a single node, this does not translate to improved performance when working with entire networks. This work is largely represented in our paper and poster, "Spike-and-Slab Probabilistic Backpropagation: When Smarter Approximations Make No Difference," (Ott and Williamson, 2022b) that was accepted at the I Can't Believe It's Not Better workshop at NeurIPS 2022.

Chapter 4 introduces an approximate method to sample from the nonparametric posterior of a normalizing flow model. We find that by sharing parameters across bootstrapped samples, we are able to perform approximate posterior inference in a computationally feasible manner, and obtain performance improvements by incorporating appropriate prior distributions. Much of what is represented in this chapter is under submission to AISTATS 2023 (Ott and Williamson, 2022a).

Chapter 5 presents a novel architecture for generative graph neural networks. Inspired by Bayesian models for sparse random graphs, this model constructs graphs edge-by-edge, in a manner that allows for graph sparsity. This work is part of a larger collaboration between myself, Curtis Carter, Elahe Ghalebi, and Sinead Williamson. I developed the results and methods included in this chapter. Elahe Ghalebi developed

related methods for link prediction as parallel work that is not presented here. Curtis Carter and Sinead Williamson acted in an advisory role.

Lastly, several appendices are included. Appendix A includes derivations and additional experimental results for Chapter 3. Appendix B includes some additional preliminary experiments performed using the model in Chapter 4 that aided in setting some model hyperparameters. Finally, Appendix C includes background information on specific topics that may be helpful to the reader that are not otherwise addressed.

Chapter 2: Background

In this dissertation, we explore topics in Bayesian inference for neural networks and in random graphs. In this chapter, we introduce concepts and related work relevant to the research presented in later chapters.

2.1 Inference Methods

A key problem in statistics is inference, that is the process of inferring the parameters of a model from data sample. In this section, we briefly outline some common approaches to statistical inference that will be used or referenced in the rest of this work.

2.1.1 Maximum Likelihood

One common approach to statistical inference is maximum (log-)likelihood estimation. Given model parameters θ and data $Y = \{y_1, \dots y_n\}$ the likelihood is $L(Y|\theta) = \mathbb{P}_{i=1}^{n} p(y_i|\theta)$. Maximizing this quantity with respect to θ yields the maximum likelihood estimate (MLE) $\hat{\theta}_{\text{MLE}}$. This estimate may be obtained in closed-form or may require the use of numerical methods like (stochastic) gradient descent.

2.1.2 Bayesian Inference

The Bayesian approach incorporates prior subjective uncertainty in the model parameters, refining that uncertainty in the presence of data through a likelihood model using Bayes' rule:

$$p(\theta|Y) = \frac{p(Y|\theta)p(\theta)}{\int p(Y|\theta)p(\theta)d\theta},$$

where $p(\theta)$ is the prior and $p(Y|\theta)$ is the likelihood. While some prior-likelihood pairs result in an analytically tractable form of the posterior $p(\theta|Y)$, many models (including neural networks) do not. Instead, we can apply methods to approximate

the posterior or to generate samples from the posterior.

2.1.2.1 Approximate Bayesian Inference

As discussed above, in many cases, the true posterior is not available in closed-form. As a result, we often turn to approximate methods. The **maximum a posteriori (MAP)** estimate yields a point estimate for θ by determining posterior mode, the value that maximizes the posterior density:

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta} p(\theta|Y).$$

Like the MLE, the MAP may be available in closed-form, or may need to be computed by numerical methods. Since the MAP is only a point-estimate, it does not provide a notion of the uncertainty about θ.

The **Laplace approximation** (Tierney and Kadane, 1986), however, uses a simple approximating distribution to the true posterior that relies on the MAP. Specifically, the Laplace approximation approximates the posterior with a Gaussian centered on the MAP with covariance matrix determined by the Hessian of the true posterior evaluated at the MAP. As such, the Laplace approximation yields an approximate posterior that is highly dependent on the behavior of the true posterior near the MAP, and may over- or under-estimate the variance and ignores multi-modality.

Variational inference (Jordan et al., 1999) introduces a "variational posterior" $q_\phi(\theta)$ which is the member of some specified family of distributions $q_\Phi = \{q_\phi|\phi \in \Phi\}$ that minimizes the Kullback-Leibler divergence with respect to the true posterior:

$$\phi^* = \arg\min_{\phi} KL(q_\phi(\cdot)\|p(\cdot|x)).$$

In practice, the KL divergence cannot be computed exactly due to the marginal likelihood (or "evidence") term $p(x)$, and instead the evidence lower bound (ELBO) is maximized, which is equivalent to minimizing the KL:

$$\phi^* = \arg\max_{\phi} \mathbb{E}_{\theta\sim q_\phi}[\log p(\theta, x)] - \mathbb{E}_{\theta\sim q_\phi}[\log q_\phi(\theta)].$$

While the choice of family for the variational posterior is highly flexible, a common approach is to use a mean-field family, where each (possibly multidimensional) parameter is treated as independent within the variational posterior:

$$q_\phi(\theta) = \prod_i q_{\phi,i}(\theta_i).$$

Assumed density filtering (ADF, Opper, 1999), also known as Bayesian online learning, is an approximate Bayesian method. ADF assumes a known parametric form of approximate posterior $q(\theta;\phi)$ and iteratively updates the parameters ϕ upon observing each new datapoint. In particular, $q(\theta;\phi^{(t)})$ is treated as a prior for datapoint y_{t+1}, yielding a new posterior

$$p(\theta|y_{t+1},\phi^{(t)}) = \frac{p(y_{t+1}|\theta)q(\theta;\phi^{(t)})}{\int p(y_{t+1}|\theta)q(\theta;\phi^{(t)})d\theta}.$$

In general, $p(\theta|y_{t+1},\phi^{(t)})$ will not be in the same distributional family as $q(\theta;\phi)$, so it will be projected into the closest distribution with parameters $\phi^{(t+1)}$. While various distributional distances could be used, the typical approach is to minimize the Kullback–Leibler divergence:

$$\phi^{(t+1)} = \arg\min_\phi KL\left(p(\cdot|y_{t+1},\phi^{(t)})\middle\|q(\cdot;\phi)\right).$$

Markov chain Monte Carlo (MCMC) is a method to sample from a Bayesian posterior, even when the posterior is not available in closed-form. MCMC algorithms produce a sequence of samples, with each sample dependent on the previous sample (hence, the "chain"). These samples can be constructed such that they will provably converge to samples from the true posterior. As such, MCMC methods are the "gold standard" for Bayesian inference methods (when the posterior is not directly available) because they converge asymptotically. However, MCMC methods are often slow to converge, requiring considerable computational resources, especially for models with a large number of parameters.

2.1.3 Nonparametric Learning

Typically, the Bayesian framework is interested in parametric models where we seek the posterior of some parameter $\theta \in \Theta \subset \mathbb{R}^p$ that indexes some probability distribution f_θ and assumes that the true data-generating distribution F_0 is contained in the model space $F_\Theta = \{f_\theta(\cdot)|\theta \in \Theta\}$. Recent work (e.g., Lyddon et al. (2018); Fong et al. (2019)) has explored a nonparametric learning (NPL) approach, where it is no longer assumed that $F_0 \in F_\Theta$. Instead, we simply consider a parameter of interest

$$\theta_0(F_0) = \arg\min_\theta \int \ell(x, \theta) dF_0(x), \tag{2.1}$$

that is the parameter of some loss function ℓ that we seek to minimize with respect to the true data-generating distribution F_0. In particular, this loss function may be the negative log-density of a parametric distribution, i.e., $-\log f_\theta(x)$. In that case, θ_0 is known to minimize $KL(f_0 \| f_\theta)$. Whereas in conventional Bayesian inference, we place a prior distribution on θ, reflecting our uncertainty in the space of the model, NPL considers placing a prior on F_0, reflecting our uncertainty about F_0 in the space of the data. This uncertainty in F_0, combined with observations $x_{1:n}$, allows us form a "nonparametric posterior" on θ as:

$$\tilde{\pi}(\theta|x_{1:n}) = \int_F \delta_{\theta_0(F)}(\theta) \pi(dF|x_{1:n}),$$

where the posterior $\pi(F|x_{1:n})$ will depend on our choice of prior $\pi(F)$.

For the choice of the Dirichlet process prior on F, following Fong et al. (2019), this gives rise to:

$$\begin{aligned}
[F|\alpha, F_\pi] &\sim \mathrm{DP}(\alpha, F_\pi) \\
[F|x_{1:n}] &\sim \mathrm{DP}(\alpha + n, G_n) \\
G_n &= \frac{\alpha}{\alpha + n} F_\pi + \frac{1}{\alpha + n} \sum_{i=1}^{n} \delta_{x_i},
\end{aligned}$$

where F_π is the base distribution, and α is the concentration parameter. Of particular note for the choice of the base distribution is the case where we have some

historical data $\tilde{x}_{1:\tilde{n}}$, which can be represented by an empirical distribution $F_\pi(x) = \frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} \delta_{\tilde{y}_i}(x)$. By sampling $F_* \sim \mathrm{DP}(\alpha + n, G_n)$, we can obtain a sample from the nonparametric posterior on θ as $\theta_* = \arg\min_\theta \int \ell(x, \theta) dF_*(x)$. In particular, because F_* will be discrete almost surely, this yields samples as:

$$\begin{aligned}
\theta(F) &= \arg\min_\theta \sum_{i=1}^{\infty} w_i \ell(s_i, \theta), \\
s_i &\overset{\text{iid}}{\sim} G_n, \quad i = 1 : \infty, \\
w_i &= \beta_i \prod_{j=1}^{i-1} (1 - \beta_j), \\
\beta_i &\sim \mathrm{Beta}(1, \alpha + n).
\end{aligned}$$

However, we can approximate this process (even for continuous base measures F_π, where an exact sample would take infinite computation) with a truncation, using a finite Dirichlet distribution for generating the weights w, using the posterior bootstrap in Algorithm 1 from Fong et al. (2019).

Algorithm 1 Posterior bootstrap sampling

Require: Observations $y_1, \ldots, y_n$, base measure F_π, number of samples B, concentration parameter α

for $b = 1, \ldots, B$ **do**

Sample m pseudo-observations $y_1^*, \ldots, y_m^* \overset{\text{iid}}{\sim} F_\pi$

Sample weights $W := (w_1, \cdots, w_n, w_1^*, \cdots, w_m^*) \sim \mathrm{Dirichlet}\left(1, \cdots, 1, \frac{\alpha}{m}, \cdots, \frac{\alpha}{m}\right)$

$F^{(b)} = \sum_{i=1}^n w_i \delta_{y_i} + \sum_{i=1}^m w_i^* \delta_{y_i^*}$

$\theta^{(b)} = \arg\min_\theta \sum_{i=1}^n w_i \ell(y_i, \theta) + \sum_{i=1}^m w_i^* \ell(y_i^*, \theta)$

end for

return $\{\theta^{(b)}\}_{b=1}^B$

2.2 Deep Learning

Neural networks (also known as artificial neural networks) are a class of highly flexible models useful for an ever-increasing number of problems involving complicated

data relationships. In this work, we will focus on feedforward neural networks, graph neural networks (GNNs, see §2.2.1), and normalizing flows. For a more comprehensive treatment of the former, see Goodfellow et al. (2016). Consider a dataset with n datapoints with covariates (or inputs) $x_i \in \mathbb{R}^p$ and responses (or outputs) $y_i \in \mathbb{R}^d$.

Feedforward neural networks (FFNNs, also known as multilayer perceptrons) are models composed of L "fully-connected" layers of the form:

$$h_\ell(x_i) = \sigma_\ell(W_\ell h_{\ell-1}(x_i) + b_\ell), \qquad \ell = 1 : L, \tag{2.2}$$

where $h_0(x_i) = x_i$ is the input to the model; W_ℓ is a $p_\ell \times p_{\ell-1}$ matrix of real-valued "weights;" b_ℓ is a p_ℓ vector of "biases" (or offsets); and σ_ℓ is an optional non-linear "activation" function, often taken to be an element-wise application of a function such as $\mathrm{ReLU}(x) = \max(x, 0)$ or $\tanh(x)$, or a transformation such as $\mathrm{softmax}(x)_i = \exp(x_i)/\sum_j \exp(x_j)$ (often used for σ_L in classification problems). A loss function is defined based on the model's output $h_L(x_i)$ and the expected response y_i, from which FFNN parameters are (primarily) estimated using backpropagation (Rumelhart et al., 1986), a clever application of the chain rule to efficiently backpropagate gradients through the network using automatic differentiation methods, updating parameter values by, for example, (stochastic) gradient descent. Generally, we will simplify notation to use only h_ℓ rather than $h_\ell(x_i)$. Additionally, appealing to the network-like structure formed by the operations to compute h_L (and the biological inspiration of neural networks), we will often refer to $(h_\ell)_j$ as a "node" or "neuron." We note that the output h_L may itself represent parameters for a likelihood function, such as the mean and (log-)variance of a Gaussian in a univariate regression problem, or as the success probabilities of a categorical distribution in a classification problem.

In practice, neural networks often incorporate many layers, whether fully-connected, recurrent, convolutional, pooling, or otherwise. The sheer number of layers of computation gives rise to the notion of a "deep neural network" or "deep learning" more generally for models that may not involve neural networks specifically.

2.2.1 Graph Neural Networks

One class of neural networks merits further discussion as background. Graph neural networks (GNNs Scarselli et al., 2008) use neural networks to incorporate graph structure into models. In this section, we review the GNN model. Here and below, graphs $G = (V, E)$ are assumed to be undirected with nodes $V = \{v_1, \ldots, v_n\}$ and edges E, unless otherwise noted. In some cases, each node or edge may have some label.

The GNN model provides for node- or graph-focused applications, for example, seeking to classify nodes in a graph or classify graphs themselves. GNN introduces a node state (referred to elsewhere as a node "embedding") x_v, meant to represent information about node v along with information in the neighborhood of v. In many ways, the GNN model is an extension of an RNN, where instead of a sequence, each node is leveraging the neighborhood graph structure. In the undirected, nonpositional graph setting, the node state is constructed from node labels ℓ_v, edge labels $\ell_{(v,u)}$ and node states in the neighborhood of v:

$$x_v = \sum_{u \in \mathcal{N}(v)} h_w(\ell_v, \ell_{(v,u)}, x_u, \ell_u), \quad v \in V,$$

where h_w is some parametric function (e.g., a FFNN). Node outputs $o_v = g_w(x_v, \ell_v)$ depend on the node state and any node labels, and may be aggregated for graph-focused applications. x_v is constructed iteratively

$$\begin{aligned} x_v(t+1) &= \sum_{u \in \mathcal{N}(v)} h_w(\ell_v, \ell_{(v,u)}, x_u(t), \ell_u), \\ o_n(t) &= g_w(x_v(t), \ell_v), \end{aligned}$$

and under mild conditions will arrive at a fixed-point. In this work, we will often refer to constructions similar to $h_w(\ell_v, \ell_{(v,u)}, x_u(t), \ell_u)$ as "messages," as we can view this iterative process as message passing between nodes. In the GNN model, the messages to node v are not directly a function of $x_v(t)$. Later work relaxes this constraint while sacrificing the contraction map guarantees of the original GNN approach.

2.2.1.1 Generative GNN Models

Of particular note for our work, several models focus on extending GNN for generative models. Here, we review deep generative models for graphs (DeepGMG Li et al., 2018b) and graph recurrent neural networks (GraphRNN You et al., 2018), although there are many such generative GNN models (e.g., Kipf and Welling (2016); Liao et al. (2019); Grover et al. (2019)).

DeepGMG relies on a node-based graph generation process, relying on three functions: $f_{\text{addnode}}(G)$, $f_{\text{addedge}}(G, v)$, and $f_{\text{node}}(G, v)$. Based on the current graph state, f_{addnode} determines whether to add an additional node or terminate the graph generation process. If a node is added, then f_{addedge} determines whether to add any edges from the newest node to any previous nodes, or move on to the next potential node. Finally, if an edge is to be added, f_{nodes} selects which of the previous nodes to connect to the new node.

Each of these three functions relies on a neural networks, and on node and graph embeddings. Node embeddings are generated using a message-passing step before the step to add a new node (optionally, also before each edge is added). With embedding h_v for node v and any edge label $x_{u,v}$ for edge (u, v), we have:

$$a_v = \sum_{u:(u,v)\in E} f_{\text{msg}}(h_u, h_v, x_{u,v})$$
$$h'_v = f_{\text{update}}(a_v, h_v),$$

where a_v is an aggregation of all messages sent to node v. In practice, the message function f_{msg} is taken to be a FFNN while the update function f_{update} is itself an RNN (specifically a GRU). The node embeddings are projected to a higher-dimensional space and aggregated to produce the graph embedding using a gating function. While our graph-generation process differs from DeepGMG, we will rely on a similar message-propagating construction to update node embeddings.

You et al. (2018) presents graph recurrent neural networks (GraphRNN). Conditioned on a node ordering π, the graph G can equivalently be expressed using the

adjacency matrix $A^\pi \in \mathbb{R}^{n\times n}$, where $A^\pi_{i,j} = \mathbb{1}\left[(\pi(v_i), \pi(v_j) \in E\right]$. Furthermore, A^π is composed into sequence S^π where

$$S^\pi_i = (A^\pi_{1,i}, A^\pi_{2,i}, \ldots, A^\pi_{i-1,i})^\top, \quad \forall i \in \{2, \ldots, n\}, \tag{2.3}$$

in other words, the adjacency vectors, with S^π_1 and S^π_{n+1} as start- and end-of-sequence tokens, respectively. GraphRNN sets out to learn the distribution of sequences, which is decomposed as

$$p\left(S^\pi\right) = \prod_{i=1}^{n+1} p(S^\pi_i | S^\pi_1, \ldots, S^\pi_{i-1}) = \prod_{i=1}^{n+1} p(S^\pi_i | S^\pi_{<i}). \tag{2.4}$$

GraphRNN uses a graph-level RNN to parameterize $p\left(S^\pi_i | S^\pi_{<i}\right)$, which is used in a node-based graph generation process. One limitation of the GraphRNN approach is its requirement of a maximum adjacency vector size. While this does not necessarily limit the total number of nodes in the graph (nodes can be constrained to only have edges to the most recent k nodes), this does pose a problem for graphs where nodes enter the graph far apart in time.

2.2.2 Normalizing Flows

Normalizing flows are, in one sense, nothing more than a complicated transformed distribution, composed of a base distribution h and a sequence of invertible transformations $g(u)$ called a "flow." Together, these construct a family of distributions p_X using a change-of-variables transformation:

$$p_X(x) = h(g^{-1}(x)) \left| \det \frac{dg^{-1}(X)}{dX} \right|_{X=x}.$$

What makes normalizing flow models interesting is the increase in flexibility of the flow due to innovations in the transformations. In particular, the flow can be composed of layers of transformations, including layers that include neural networks (see discussion below). As a result, this yields a family of resultant transformations – resembling what one might expect from neural networks generally – useful for a

variety of applications including density estimation, while often enforcing a diagonal Jacobian matrix to maintain efficient computation of the log-likelihood. Kobyzev et al. (2021) provides an introduction and review of normalizing flows, but a few particular models and innovations are discussed below.

Coupling layers are able to incorporate highly flexible transformations on a subset of the dimensions. For example, let $x_1, x_2, u_1, u_2 \in \mathbb{R}^d$ with $x = [x_1; x_2]$ and $u = [u_1; u_2]$. We can include a non-invertible transformation (such as a neural network) $g : \mathbb{R}^d \to \mathbb{R}^d$ by applying the identity transformation the first subset of dimensions:

$$\begin{aligned} x &= f^{-1}(u) \\ x_1 &= u_1 \\ x_2 &= u_2 + g(u_1) \end{aligned}$$

. This yields an invertible transformation of $f^{-1}(u)$ such that $f(x) = [x_1; x_2 - g(x_1)]$. In fact, for this example, the Jacobian is simply 1, yielding what is known as a "volume-preserving" flow. We can use many such coupling layers (alternating which components have the identity transformation applied) to construct highly-flexible flows.

Non-linear independent components estimation (NICE, Dinh et al., 2014) applies this coupling idea to allow for a (typically) non-invertible neural network step as part of a transformation, but only on one block of the data at a time. This is easily inverted: one block is the input to the network, and copied to the output block. NICE uses additive coupling layers to avoid computing the Jacobian determinant, but allows for a single scaling layer which gives a constant term in the log-likelihood.

Density estimation using real NVP (Dinh et al., 2016) follows up to NICE and provides a helpful comparison to VAEs, namely that VAEs use an approximate inference network (the encoder mapping x to z). This work allows for high-dimensional highly nonlinear bijections that remain computationally efficient. Here, we simply

recognize that the determinant of a triangular matrix is simply the product of the diagonal elements. They now use *affine* coupling layers (rather than simply additive), allowing for scaling, creating a shift-and-scale: $y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d})$, where $\odot$ is element-wise product. Importantly, the diagonal elements of the Jacobian matrix are simple and do not require the Jacobian of s or t (obviously, to maximize the log-likelihood, there will be gradients of s and t, but we need not deal with the Hessian), and neither s nor t need to be themselves invertible.

The masked autoregressive flow (MAF, Papamakarios et al., 2017) constructs a normalizing flow for density estimation in an autoregressive model. In particular, it includes layers built from masked autoencoder for distribution estimation (MADE, Germain et al., 2015), conditioning each component of the output x_i in a particular layer on the previous variables $x_{1:i-1}$.

2.3 Bayesian Neural Networks

Typically, inference in neural network models is performed by applying stochastic gradient descent in order to maximize the log-likelihood of the data (or to minimize some other loss function). As discussed above, this ignores inherent uncertainty in model parameters, leading many researchers to apply Bayesian inference methods to neural networks. In this section, we describe common approaches to Bayesian neural networks.

Bayesian neural networks (MacKay, 1992b; Neal, 1995), in an effort to quantify uncertainty in the neural network weights (and biases), apply a Bayesian approach, incorporating a prior on the weights with an explicit likelihood (rather than a more general loss function). Typically, this prior is assumed to be Gaussian. The posterior is generally not available in closed-form, so MCMC (Neal, 1995) or approximate methods like variational inference (Graves, 2011; Blundell et al., 2015) are typically used. The approximate posterior (or samples from the posterior in the case of MCMC) can then be used for the posterior predictive distribution or any other tasks.

2.3.1 Probabilistic Backpropagation

A particular approximate Bayesian method for neural networks relevant to our work is probabilistic backpropagation (PBP, Hernández-Lobato and Adams, 2015). PBP assumes an FFNN architecture (with input scaling at each layer) with ReLU activation functions in a regression setting, an independent Gaussian prior on every element of the weight matrices (including bias vectors by augmenting the input to each layer with a constant 1), and with Gamma priors on the precision for the weight priors and likelihood:

$$\begin{aligned} p(\mathbf{y}|\mathcal{W}, \mathbf{X}, \gamma) &= \prod_{n=1}^{N} N(y_n|f(\mathbf{x}_n; \mathcal{W}), \gamma^{-}1) \\ p(\mathcal{W}|\lambda) &= \prod_{\ell=1}^{L} \prod_{i=1}^{n_\ell} \prod_{j=1}^{n_{\ell-1}+1} N(w_{ij,\ell}|0, \lambda^{-}1) \\ p(\lambda) &= \text{Gamma}(\lambda|\alpha_0^\lambda, \beta_0^\lambda) \\ p(\gamma) &= \text{Gamma}(\gamma|\alpha_0^\gamma, \beta_0^\gamma), \end{aligned}$$

where $\mathcal{W}$ concatenates the weight matrices in all ℓ layers, n_ℓ is the number of nodes in layer ℓ, and $\alpha_0^\lambda, \beta_0^\lambda, \alpha_0^\gamma, \beta_0^\gamma$ are fixed hyperparameters.

The true posterior on the weights is not available in closed-form. Instead, PBP assumes a mean-field Gaussian approximate posterior on the weights, with independent Gamma approximate posteriors for the two precision parameters:

$$q(\mathcal{W}, \gamma, \lambda) = \left(\prod_{\ell=1}^{L} \prod_{i=1}^{n_\ell} \prod_{j=1}^{n_{\ell-1}+1} N(w_{ij,\ell}|m_{ij,\ell}, v_{ij,\ell}) \right) \text{Gamma}(\lambda|\alpha^\lambda, \beta^\lambda)\text{Gamma}(\gamma|\alpha^\gamma, \beta^\gamma),$$

where $m_{ij,\ell}$ and $v_{ij,\ell}$ are the approximate posterior's mean and variance of weight $w_{ij,\ell}$ (for simplicity, we omit discussion here about parameters λ and γ, though Hernández-Lobato and Adams (2015) gives a full treatment). To learn this approximate posterior q, PBP uses assumed density filtering (see Opper (1999) and discussion above in §2.1.2.1), relying on specific properties of Gaussian distributions (Minka, 2001) that result in a method resembling backpropagation. Specifically, with the current beliefs

regarding weight w as expressed by the approximate posterior being $q(w; m, v) = N(w|m, v)$, we note that the Bayesian way to update those beliefs upon seeing a new datapoint $(\mathbf{x}_*, y_*)$ would be expressed by a distribution $s(w) = f(\mathbf{x}_*, y_*|w)q(w)Z^{-1}$ where Z^{-1} is the normalization constant and f represents the likelihood.

In the ADF setting, we select the distribution q^{new} in the approximate posterior family that minimizes $KL(s\|q^{\text{new}})$. In general, this is highly non-trivial, but PBP makes note that the solution for the Gaussian family has a gradient-based update,

$$m^{\text{new}} = m + v\frac{\partial \log Z}{\partial m} \qquad v^{\text{new}} = v - v^2\left[\left(\frac{\partial \log Z}{\partial m}\right)^2 - 2\frac{\partial \log Z}{\partial v}\right].$$

While the normalization constant is not available in closed-form, PBP makes several simplifying assumptions to develop a closed-form approximation. Specifically, PBP makes the following assumption for the form of the normalization constant for datapoint $(\mathbf{x}_*, y_*)$:

$$\begin{aligned} Z &= \int N(y_*|f(\mathbf{x}_*|\mathcal{W}), \gamma^{-1})q(\mathcal{W}, \gamma, \lambda)d\mathcal{W}d\gamma d\lambda \\ &\approx N(y_*|m^{z_L}, \beta^\gamma/(\alpha^\gamma - 1) + v^{z_L}), \end{aligned}$$

where m^{z_L} and v^{z_L} are the mean and variance of the neural network's output layer, z_L, under the approximate posterior. The construction of the network is composed of the steps:

$$\begin{aligned} z_0 &= \mathbf{x}_*, \\ a_\ell &= W_\ell z_{\ell-1}/\sqrt{n_{\ell-1} + 1}, \quad \ell = 1 : L, \\ b_\ell &= \text{ReLU}(a_\ell), \quad \ell = 1 : L - 1, \\ z_\ell &= [b_\ell; 1], \quad \ell = 1 : L - 1, \end{aligned}$$

with the output $z_L = a_L$. PBP proposes a layer-by-layer moment-matching procedure whereby the uncertainty of weights is propagated by approximating the distributions for a_ℓ, b_ℓ, z_ℓ with diagonal multivariate Gaussians, such that

$$z_\ell \sim N(m^{z_\ell}, v^{z_\ell}) \qquad a_\ell \sim N(m^{a_\ell}, v^{a_\ell}) \qquad b_\ell \sim N(m^{b_\ell}, v^{b_\ell})$$

with the following forms:

$$
\begin{aligned}
m^{a_\ell} &= M_\ell m^{z_{\ell-1}}/\sqrt{n_{\ell-1}+1}, \\
v^{a_\ell} &= [(M_\ell \circ M_\ell)v^{z_{\ell-1}} + V_\ell(m^{z_{\ell-1}} \circ m^{z_{\ell-1}}) + V_\ell v^{z_{\ell-1}}]/(n_{\ell-1}+1), \\
m_i^{b_\ell} &= \Phi(\alpha_i)v_i', \\
v_i^{b_\ell} &= m_i^{b_\ell} v_i' \Phi(-\alpha_i) + \Phi(\alpha_i)v_i^{a_\ell}(1-\gamma_i(\gamma_i+\alpha_i)), \\
v_i' &= m_i^{a_\ell} + \sqrt{v_i^{a_\ell}}\gamma_i, \\
\alpha_i &= \frac{m_i^{a_\ell}}{\sqrt{v_i^{a_\ell}}}, \\
\gamma_i &= \frac{\phi(-\alpha_i)}{\Phi(\alpha_i}, \\
m^{z_\ell} &= [m^{b_\ell}; 1], \\
v^{z_\ell} &= [v^{z_\ell}; 0],
\end{aligned}
$$

where ϕ and Φ are the pdf and cdf of the standard Gaussian, and M_ℓ and V_ℓ are matrices with $q(w_{ij,\ell}) = N(w_{ij,\ell}|M_{ij,\ell}, V_{ij,\ell})$.

With this approximation to the mean and variance of the output layer z_ℓ, we can approximate the normalizing constant Z and update the mean and variance of the approximate posterior for each weight, as above. As such, with a gradient-based update in hand for the approximate posterior, automatic differentiation software can be employed. In practice, PBP also includes an expectation propagation step after each pass through the data, although that will not be the focus of our work in Chapter 3.

Chapter 3: Spike-and-Slab Probabilistic Backpropagation[1]

Abstract

Probabilistic backpropagation (PBP, Hernández-Lobato and Adams, 2015) is an approximate Bayesian inference method for deep neural networks, using a message-passing framework. These messages—which correspond to distributions arising as we propagate our input through a probabilistic neural network—are approximated as Gaussian. However, in practice, the exact distributions may be highly non-Gaussian. In this paper, we propose a more realistic approximation based on a spike-and-slab distribution. Unfortunately, in this case, better approximation of the messages does not translate to better downstream performance. We present results comparing the two schemes and discuss why we do not see a benefit from this spike-and-slab approach.

3.1 Introduction

Deep neural networks are flexible non-linear models common to domains with complicated data relationships. However, despite having many unknown model parameters, many neural networks do not attempt to represent model (epistemic) or data (aleatoric) uncertainty. Bayesian approaches to neural networks could capture

[1]This chapter will appear as: Evan Ott and Sinead Williamson. Spike-and-Slab Probabilistic Backpropagation: When Smarter Approximations Make No Difference. In *I Can't Believe It's Not Better Workshop: Understanding Deep Learning Through Empirical Falsification*, 2022b. URL `https://openreview.net/forum?id=iYAdBHSA_Pt`.

this uncertainty but are typically rendered intractable in closed-form. MCMC-based methods (Neal, 1995; Cobb and Jalaian, 2021) offer asymptotic guarantees but are typically slow to converge. Instead, it is typical to use approximate inference algorithms, such as those based on Laplace approximations (MacKay, 1992a; Daxberger et al., 2021) or variational inference (Graves, 2011; Blundell et al., 2015).

Probabilistic backpropagation (PBP, Hernández-Lobato and Adams, 2015) is an example of an approximation-based approach for Bayesian deep learning. Like many such approaches, it uses a mean-field approximation to the posterior, which is inferred using a message-passing algorithm based on assumed density filtering (Opper, 1999). The resulting algorithm is reminiscent of backpropagation; however, rather than propagating a single function estimate through a neural network, PBP propagates distributions representing our estimated posterior uncertainty. PBP approximates these distributions, or "messages," using a Gaussian, whose parameters are a known function of the means and variances of the per-weight distributions, allowing us to update those weight parameters using gradient information in a backwards pass.

In practice, the true messages can be highly non-Gaussian, since they have been propagated through a ReLU or similar nonlinearity. Using a Gaussian approximation risks ignoring sparsity inherent in the true message structure. In this paper, we show that the PBP algorithm can be modified to use sparse messages, parameterized using a spike-and-slab distribution. Computational costs are not significantly increased over Gaussian messages.

Unfortunately, we discover what many before us have discovered in different contexts: Gaussian approximations usually work pretty well. Closer investigation shows that, if we use bias terms in our neural network (as is typical), any theoretical advantage of our spike-and-slab approach evaporates. In the absence of a bias term, our approximation does differ from the standard PBP approach, and at a per-message level does indeed provide better approximations to the true message. However, this advantage does not translate into a significant difference in overall

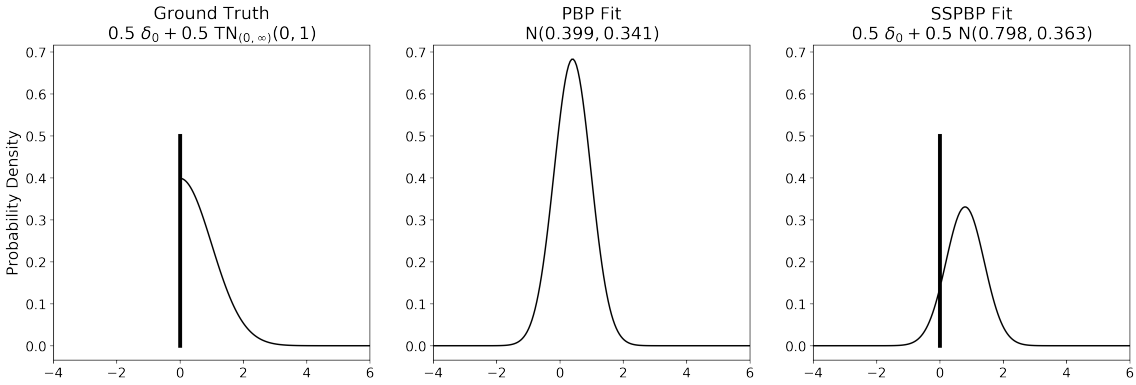

Figure 3.1: *Left*, the true distribution obtained by applying a ReLU to a $\mathrm{N}(0, 1)$ random variable, with a mixture of a truncated Gaussian and a "spike" (with its probability indicated with an overlayed vertical bar) at $X = 0$. *Center*, the Gaussian density obtained in the PBP approximation. *Right*, the spike-and-slab distribution obtained by our approximation (with the spike probability indicated with an overlayed vertical bar).

performance. While this is a negative result, and while it remains possible that alternative approximations could yield improved performance, our work indirectly highlights the fact that Gaussian approximations to non-Gaussian distributions are often a good choice in practice.

3.2 Probabilistic Backpropagation

We consider the setting of feed-forward neural networks (FFNNs) with ReLU activation $\sigma(x) = \max(x, 0)$, such that

$$\begin{aligned} \mathbf{z}^{(\ell)}(\mathbf{x}) =& \sigma\left(\mathbf{W}^{(\ell)}\left[\mathbf{z}^{(\ell-1)}(\mathbf{x}); 1\right] \Big/ \sqrt{n_{\ell-1}+1}\right), \quad \ell = 1 : L-1 \\ \hat{\mathbf{y}} =& \mathbf{W}^{(L)}\left[\mathbf{z}^{(L-1)}(\mathbf{x}); 1\right] / \sqrt{n_{L-1}+1}\,, \end{aligned} \tag{3.1}$$

with $[a; b]$ indicating concatenation, n_ℓ indicating the number of nodes in layer ℓ, and $\mathbf{z}^{(0)}(\mathbf{x}) = \mathbf{x}$. Let $\mathcal{W} = (\mathbf{W}^{(\ell)})_{\ell=1}^{L}$ where $\mathbf{W}^{(\ell)} = (w_{ij}^{(\ell)})_{i=1}^{n_{\ell-1}+1}{}_{j=1}^{n_\ell}$ be the set of all weights (which includes the biases).[2] We include a scaling factor of $\sqrt{n_{\ell-1}+1}$ to make the

[2]We adopt the convention throughout that x indicates a scalar; $\mathbf{x}$ a vector; $\mathbf{X}$ a matrix; and $\mathcal{X}$ a list of matrices.

scale of the input to each neuron invariant to the size of the previous layer, following Hernández-Lobato and Adams (2015).

To make the model Bayesian, we place a spherical Gaussian prior on the weights, meaning the $\mathbf{z}^{(\ell)}(\mathbf{x})$ and $\hat{\mathbf{y}}$ are now random, and we assume $y_i \sim \mathrm{N}(\hat{y}_i, \gamma^{-1})$. Since the number of weights can be large, and the ReLU nonlinearity removes the possibility of analytic tractability, posterior inference can be computationally challenging.

Probabilistic backpropagation (PBP, Hernández-Lobato and Adams, 2015) is an approximate inference technique appropriate for this setting. PBP uses assumed density filtering (ADF, Opper, 1999) to update a mean-field approximation to the true posterior,[3]

$$q(\mathcal{W}) = \prod_{\ell} q^{(\ell)}(\mathbf{W}^{(\ell)}) = \prod_{i,j,\ell} \mathrm{N}(w_{ij}^{(\ell)} | m_{ij}^{(\ell)}, v_{ij}^{(\ell)}). \tag{3.2}$$

The mean and variance parameters of the weights ($m_{ij}^{(\ell)}$ and $v_{ij}^{(\ell)}$) are updated by propagating uncertainty through the network, that is, by evaluating the distributions over the representations $\mathbf{z}^{(\ell)}(\mathbf{x})$. PBP approximates these distributions with Gaussian distributions, obtained through moment matching. Leveraging some properties of Gaussian distributions (Minka, 2001), PBP is able to update the parameters of the approximate posterior in an approach analogous to backpropagation. Finally, PBP also incorporates an expectation propagation step to further refine the approximate posterior after each full pass through the training data.

Several extensions to the PBP framework have provided improvements, such as including approximations appropriate for classification (Ghosh et al., 2016), incorporating minibatching (Benatan and Pyzer-Knapp, 2018), and using non-diagonal Gaussians as the approximating distribution (Sun et al., 2017). Similar approaches have been used in a variational inference context (Roth and Pernkopf, 2016; Wu et al.,

[3]We ignore here terms concerning the observation noise variance and the prior variance of the Gaussians.

2018; Dera et al., 2019; Haußmann et al., 2020) and in a hybrid Bayesian/maximum likelihood context (Gast and Roth, 2018).

3.2.1 Limitations of a Gaussian Approximation

PBP approximates the distribution of $\mathbf{z}^{(\ell)}(\mathbf{x})$—that is, the "messages" in the ADF algorithm—using Gaussians. However, the ReLU nonlinearity in Equation 3.1 means that the true distributions can be highly non-Gaussian. As a simple example, consider the distribution over the jth element of the first-layer representation of an input $\mathbf{x}$, $z_j^{(1)}(\mathbf{x}) = \sigma({\mathbf{w}_j^{(1)}}^\top[\mathbf{x}; 1])$. We are approximating the posterior distributions over $w_{ij}^{(1)}$ using Gaussians, meaning that the distribution implied by $\mathbf{w}_j^\top[\mathbf{x}; 1]$ is also Gaussian. However, once this has passed through a ReLU, the distribution over $z_j^{(1)}(\mathbf{x})$ is a mixture of a Dirac delta distribution (or "spike") at $x = 0$ and a Gaussian truncated to the domain $(0, \infty)$. See Figure 3.1 for a demonstration.

3.3 Spike-and-Slab Probabilistic Backpropagation

An exact implementation of ADF in the FFNN would send messages based on the true distribution over the $\mathbf{z}^{(\ell)}(\mathbf{x})$, given the current approximating distribution $q(\mathcal{W})$. We will refer to this true distribution as $q(\mathbf{z}^{(\ell)})$[4]. This is computationally infeasible: while we can calculate this distribution for a single layer where the input is the observed x (as shown above), on later layers the inputs are themselves the propagated distributions. PBP chooses to approximate these with diagonal Gaussian distributions $\widetilde{q}_{\text{PBP}}(\mathbf{z}^{(\ell)})$, obtained through moment matching.

We propose using a more sophisticated approximation of the messages $q(\mathbf{z}^{(\ell)})$. At the first layer, we know that $q(\mathbf{z}^{(1)})$ is a spike-and-slab distribution with truncated Gaussian slab. We choose to model the resulting sparsity directly using a spike-and-slab distribution with a (non-truncated) Gaussian slab, such that for node i in layer

[4]For notational conciseness, we hereafter write $\mathbf{z}^{(\ell)}$ in place of $\mathbf{z}^{(\ell)}(\mathbf{x})$

ℓ, we have

$$\widetilde{q}_{\text{SSPBP}}(z_i^{(\ell)}) = (1 - \widetilde{\rho}_i^{(\ell)})\delta_0(z_i^{(\ell)}) + \widetilde{\rho}_i^{(\ell)}\text{N}(z_i^{(\ell)}|\widetilde{m}_i^{(\ell)}, \widetilde{v}_i^{(\ell)}), \tag{3.3}$$

with slab probability $\widetilde{\rho}_i^{(\ell)}$, slab mean $\widetilde{m}_i^{(\ell)}$, and slab variance $\widetilde{v}_i^{(\ell)}$. We refer to the resulting algorithm as Spike-and-Slab PBP (SSPBP).

3.3.1 Approximating Messages Using a Spike-and-Slab

We wish to approximate some distribution p, with a spike-and-slab distribution q of the form of Equation 3.3. We can do so by minimizing the Kullback-Leibler divergence $\text{KL}(p\|q)$, yielding the following parameters (see Appendix A.1.1 for derivation):

$$\rho = \mathbb{P}_{X\sim p}[X \neq 0], \qquad m = \frac{1}{\rho}\mathbb{E}_{X\sim p}[X], \qquad v = \frac{1}{\rho}\left(\mathbb{V}_{X\sim p}[X] - \rho(1-\rho)m^2\right).$$

Note that this is similar to the moment-matching setting of PBP: we first match the spike probability $(1-\rho)$, and then match the first and second moments.

Consider the jth node of the ℓth layer of our FFNN. Following Hernández-Lobato and Adams (2015), we augment the incoming message with a bias term, so that $\widetilde{\rho}_{n_\ell+1}^{(\ell)} = 1$, $\widetilde{m}_{n_\ell+1}^{(\ell)} = 1$, and $\widetilde{v}_{n_\ell+1}^{(\ell)} = 0$. Let $\mathbf{M}^{(\ell)} = (\mathbf{m}_i^{(\ell)})_{i=1}^{n_\ell}$ and $\mathbf{m}_i^{(\ell)} = (m_{ij}^{(\ell)})_{j=1}^{n_{\ell-1}+1}$ be the means of the approximate posteriors $q(w_{ij}^{(\ell)})$ of the weights of the FFNN, and let $\mathbf{V}^{(\ell)} = (\mathbf{v}_i^{(\ell)})_{i=1}^{n_\ell}$ and $\mathbf{v}_i^{(\ell)} = (v_{ij}^{(\ell)})_{j=1}^{n_{\ell-1}+1}$ be the corresponding variances (Equation 3.2). In the absence of a nonlinearity (e.g., for a final regression layer), we approximate the message $z_j^{(\ell)}$ with a spike-and-slab distribution, with slab probability $\widetilde{\rho}_j^{(\ell,\text{linear})}$, slab mean $\widetilde{m}_j^{(\ell,\text{linear})}$, and slab variance $\widetilde{v}_j^{(\ell,\text{linear})}$ where

$$\widetilde{\rho}_j^{(\ell,\text{linear})} \equiv \widetilde{\rho}^{(\ell,\text{linear})} = 1 - \prod_{i=1}^{n_{\ell-1}+1}(1 - \widetilde{\rho}_i^{(\ell-1)}), \qquad \widetilde{m}_j^{(\ell,\text{linear})} = \frac{\mathbf{m}_j^{(\ell)\top}\left(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)}\right)}{\widetilde{\rho}^{(\ell,\text{linear})}\sqrt{n_{\ell-1}+1}}, \tag{3.4}$$

$$\widetilde{v}_j^{(\ell,\text{linear})} = \frac{c_i}{(n_{\ell-1}+1)\,\widetilde{\rho}^{(\ell,\text{linear})}} - \left(1 - \widetilde{\rho}^{(\ell,\text{linear})}\right)\left(\widetilde{m}_j^{(\ell,\text{linear})}\right)^2,$$

where $\circ$ indicates the element-wise Hadamard product and

$$\begin{aligned}\mathbf{c} =& \mathbf{V}^{(\ell)}\left(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)}\right) + \left(\mathbf{M}^{(\ell)} \circ \mathbf{M}^{(\ell)}\right)\left(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \widetilde{\mathbf{v}}^{(\ell-1)}\right) \\ &+ \mathbf{V}^{(\ell)}\left(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \widetilde{\mathbf{v}}^{(\ell-1)}\right) + \left(\mathbf{M}^{(\ell)} \circ \mathbf{M}^{(\ell)}\right)\left(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \left(1 - \widetilde{\boldsymbol{\rho}}^{(\ell-1)}\right) \circ \widetilde{\mathbf{m}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)}\right).\end{aligned}$$

If we pass this through a ReLU, we have

$$\widetilde{\rho}_j^{(\ell,\text{ReLU})} = \widetilde{\rho}^{(\ell,\text{linear})}\Phi\left(\alpha_j\right), \qquad \widetilde{m}_j^{(\ell,\text{ReLU})} = \widetilde{m}_j^{(\ell,\text{linear})} + \gamma_j\sqrt{\widetilde{v}_j^{(\ell,\text{linear})}},$$

$$\widetilde{v}_j^{(\ell,\text{ReLU})} = \left(1 - \gamma_j\alpha_j - \gamma_j^2\right)\widetilde{v}_j^{(\ell,\text{linear})},$$

with intermediate values $\alpha_j = \widetilde{m}_j^{(\ell,\text{linear})}/\sqrt{\widetilde{v}_j^{(\ell,\text{linear})}}$ and $\gamma_j = \phi\left(\alpha_j\right)/\Phi\left(\alpha_j\right)$, where ϕ and Φ are the PDF and CDF of a standard Gaussian, respectively. Derivations are provided in Appendix A.1.

3.4 Empirical Analysis

We begin by numerically validating our approximation before exploring how well it performs when used in a Bayesian FFNN. Code can be found at the anonymous repository `https://github.com/SSPBP/SSPBPcode`. All hyperparameters follow those used by Hernández-Lobato and Adams (2015) unless otherwise stated.

3.4.1 Quality of the Spike-and-Slab Approximation

To explore why the spike-and-slab approximation should yield a better representation of the propagated probability distributions than a Gaussian, we performed a simulation study. Here, we explore a ReLU transformation $T = \text{ReLU}(XW)$ applied to the product of two independent Gaussians X and W. We apply the PBP and SSPBP approximations to determine how the two approaches would approximate the resultant distribution, then we compare 10,000 samples of the approximate distributions to a ground truth sample by computing the maximum mean discrepancy (MMD) with a squared exponential kernel $K(x, y) = \exp(-\gamma(x - y)^2)$, setting $\gamma = 1$.

The results are summarized in Table 3.1. The first two columns show the parameters of the underlying distributions, and the third column shows the percent of the distribution that is saturated to zero by the ReLU nonlinearity. The fourth column shows the MMD between two size-10,000 samples from the true distribution, to give an idea of the scale of the differences. Next, we see the MMD between the PBP approximation and the true distribution (column 5) and the MMD between our SSPBP approximation and the true distribution (column 6). We see from these results that, in cases where the true distribution has non-trivial sparsity, samples from SSPBP more closely match the true distribution compared with PBP, giving credence to the intuition that a spike-and-slab approximation should improve the PBP framework.

p_X	p_W	% Saturated	Same	PBP	SSPBP
$\mathrm{N}(0,1)$	$\mathrm{N}(0,1)$	49.86%	0.000028	0.066	**0.020**
$\mathrm{N}(1,1)$	$\mathrm{N}(3,1)$	16.24%	0.000090	0.031	**0.015**
$\mathrm{N}(1,1)$	$\mathrm{N}(-3,1)$	84.44%	0.000055	0.21	**0.0038**
$\mathrm{N}(3,1)$	$\mathrm{N}(3,1)$	0.22%	0.00025	**0.0043**	0.0051
$\mathrm{N}(3,1)$	$\mathrm{N}(-3,1)$	99.72%	0.00000058	0.017	**0.00024**

Table 3.1: Simulation study of how well SSPBP approximates the true distribution after a single layer, reporting the MMD between a ground truth sample and approximations obtained using either PBP or SSPBP, along with the MMD between two ground truth samples.

3.4.2 Evaluation as Part of a Bayesian FFNN

Since our spike-and-slab approximation mimics the sparsity inherent in the FFNN, we expected it to achieve better performance. To evaluate this, we compared our approach to PBP, modifying the existing official Theano (Theano Development Team, 2016) implementation, which was released under a BSD 3-Clause "New" or "Revised" License.[5] We applied several different model architectures to eight regression datasets used in Hernández-Lobato and Adams (2015). We trained the models

[5] `https://github.com/HIPS/Probabilistic-Backpropagation`

using 40 training epochs (without early stopping), with an 80%-20% training-test split, repeating the process five times for each dataset-model comparison (following the original PBP settings). We report the mean and standard error of the average test set RMSE across those five repetitions in Table 3.3. Running all experiments in Tables 3.3 and 3.4 took under 12 hours on an Intel i5 4 core 3.2 GHz desktop, using CPU computation.

As Table 3.3 indicates, the performance is comparable across the two models. Disappointingly, there is no significant difference between the two approaches.

A hint at why this is can be seen by considering the estimate of the spike probability in Equation 3.4. Our approximation *only* captures spikes at zero. However, if our FFNN includes bias terms, this bias will translate spikes at zero in $\mathbf{z}^{(\ell-1)}$ to non-zero locations in $\mathbf{z}^{(\ell)}$, meaning our approximation will not capture them. This means all mass is placed on the slab, rendering our approximation *identical* to modeling solely with a Gaussian as in the original PBP framework. We show this formally in Appendix A.2. The variation in Table 3.3 is likely due to numerical instability, using different random seeds, or other similar issues, not as a result of improving the internal approximations of the PBP framework.

The algorithms *do* however differ if we do not include a bias term. In Table 3.4 we repeat our experiments without a bias term. Unfortunately, despite the fact that we are now indeed comparing different update rules, there remains no significant difference between the algorithms.

We hypothesise that this may be because the slab probabilities $\widetilde{\rho}^{(\ell,\text{linear})}$ would tend toward one given that the product in Equation 3.4 will tend toward zero in all but the most extreme cases. To test this, we compute $\widetilde{\rho}^{(\ell,\text{linear})}$ for the linear layers in several architectures trained on the Boston dataset. See Table 3.2 for results. Note that $\widetilde{\rho}^{(1,\text{linear})}$ is always one by construction, since the signal has not passed through any nonlinearities. For later layers, even though the signal *has* passed through nonlinearities, we see that the slab probability does tend towards one in practice, even

for narrow networks.

Table 3.2: Mean and standard error of the average slab probability in linear layers, $\widetilde{\rho}^{(\ell,\text{linear})}$, on test set observations for various architectures on the Boston dataset. 30 trials were completed for each choice of hidden layers.
† Some trials removed due to numerical instabilities.

Hidden Layers	$\widetilde{\rho}^{(1,\text{linear})}$	$\widetilde{\rho}^{(2,\text{linear})}$	$\widetilde{\rho}^{(3,\text{linear})}$	Output $\hat{y}$
5	1.000 ± 0.000	–	–	0.976 ± 0.007
50	1.000 ± 0.000	–	–	1.000 ± 0.000
5×5	1.000 ± 0.000	0.962 ± 0.007	–	$0.968 \pm 0.006^{\dagger}$
50×50	1.000 ± 0.000	1.000 ± 0.000	–	1.000 ± 0.000
$5 \times 5 \times 5$	1.000 ± 0.000	0.958 ± 0.008	$0.970 \pm 0.008^{\dagger}$	$0.971 \pm 0.009^{\dagger}$
$50 \times 50 \times 50$	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

3.5 Related Work[6]

Perhaps the most popular BNN approach is variational inference. This approach supposes a family of distributions for the (variational) posterior and selects the particular distribution with the lowest Kullbeck-Leibler (KL) divergence to the true posterior by way of optimizing the evidence lower bound (ELBO). Variational inference has several benefits, including leveraging existing automatic differentiation tools. Existing work (Graves, 2011; Blundell et al., 2015) has applied variational inference to Bayesian neural networks. However, in this setting, samples of the variational posterior are required to propagate through the network, which can be computationally expensive.

Recent work has explored the use of a spike-and-slab prior in related contexts. Bai et al. (2019, 2020) employ a spike-and-slab prior and variational posterior family in the context of neural networks, but again, they require samples from the posterior to assess the uncertainty in the posterior predictive distribution. Sun et al. (2022) explored the use of stochastic gradient Hamiltonian Monte Carlo for a spike-and-slab

[6]This section does not appear in Ott and Williamson (2022b).

	1 Layer 50 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**3.554±0.179**	3.566±0.198
Combined Cycle Power Plant	4.117±0.037	**4.116±0.033**
Concrete Compression Strength	**5.616±0.125**	5.662±0.105
Energy Efficiency	1.857±0.081	**1.856±0.079**
Kin8nm	**0.098±0.001**	0.100±0.001
Naval Propulsion	**0.006±0.000**	**0.006±0.000**
Wine Quality Red	0.655±0.003	**0.654±0.003**
Yacht Hydrodynamics	**1.344±0.061**	1.367±0.071
	2 Layers 10 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	3.097±0.147	**2.997±0.165**
Combined Cycle Power Plant	**4.088±0.067**	4.096±0.066
Concrete Compression Strength	6.031±0.161	**5.921±0.158**
Energy Efficiency	**1.477±0.043**	1.660±0.112
Kin8nm	0.111±0.004	**0.109±0.002**
Naval Propulsion	**0.006±0.000**	**0.006±0.000**
Wine Quality Red	0.653±0.012	**0.652±0.008**
Yacht Hydrodynamics	**1.064±0.072**	1.131±0.063

Table 3.3: Mean and standard error of average test set RMSE of PBP and SSPBP, on eight datasets.

prior in a BNN setting. Fang et al. (2020) uses stochastic expectation propagation (Li et al., 2015), an inference method similar to the one used in PBP, with spike-and-slab prior and approximate posterior, but in a simpler probit regression setting.

3.6 Discussion

In this paper, we presented SSPBP, a spike-and-slab variant of probabilistic backpropagation. Ultimately, we determine that, empirically and analytically, the use of a spike-and-slab approximation does not improve performance, despite seeming to

be a more intuitive approximation for the problem. While using a spike-and-slab approximate posterior distribution for model parameters could—in theory—provide better results in a bias-free FFNN, our investigation of this setting casts doubt on this intuition: the spike-and-slab approximation *is* able to model sparsity inherent to the ReLU activation function but fails to produce better empirical results, likely due to the fact that, in practice, the slab probability tends to saturate towards one. Possible future directions could include assessing alternative approximations or incorporating spike-and-slab approximations in formulations of approximate Bayesian inference like variational inference; however, it appears that a Gaussian approximation is hard to beat.

	1 Layer 50 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**3.529±0.296**	3.544±0.301
Combined Cycle Power Plant	4.300±0.054	**4.295±0.052**
Concrete Compression Strength	7.092±0.108	**7.010±0.156**
Energy Efficiency	**1.788±0.038**	1.789±0.039
Kin8nm	0.159±0.002	**0.158±0.002**
Naval Propulsion	**0.006±0.000**	**0.006±0.000**
Wine Quality Red	0.621±0.013	**0.618±0.013**
Yacht Hydrodynamics	**6.200±0.269**	6.319±0.282
	2 Layers 10 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**3.809±0.295**	3.865±0.322
Combined Cycle Power Plant	4.190±0.017	**4.188±0.023**
Concrete Compression Strength	6.823±0.214	**6.668±0.237**
Energy Efficiency	1.699±0.041	**1.617±0.019**
Kin8nm	0.126±0.001	**0.125±0.001**[†]
Naval Propulsion	**0.005±0.000**	0.006±0.000
Wine Quality Red	0.635±0.011	**0.633±0.014**
Yacht Hydrodynamics	**3.898±0.245**	4.276±0.390

Table 3.4: Mean and standard error of average test set RMSE for the bias-free versions of PBP and SSPBP, on eight datasets.
[†] Due to numerical issues, the trials for this model were repeated with a different random seed.

Chapter 4: Nonparametric Posterior Normalizing Flows[1]

Abstract

Normalizing flows allow us to construct complex probability distributions from simpler distributions using a change-of-variables transformation. If we model this transformation using an invertible neural network with an analytically tractable Jacobian, we can train that neural network to perform maximum likelihood density estimation (Dinh et al., 2014).

Such maximum likelihood density estimation is likely to overfit, particularly if the number of observations is small. Traditional Bayesian approaches offer the prospect of capturing posterior uncertainty, but come at high computational cost and do not provide an intuitive way of incorporating prior information. A nonparametric learning approach (Lyddon et al., 2018) allows us to combine observed data with priors on the space of observations. We present a scalable approximate inference algorithm for nonparametric posterior normalizing flows, and show that the resulting distributions can yield improved generalization and uncertainty quantification.

[1]Some work presented here is currently under review as part of: Evan Ott and Sinead Williamson. Nonparametric Posterior Normalizing Flows. submitted, 2022a.

4.1 Introduction[2]

Normalizing flows (Tabak and Vanden-Eijnden, 2010; Dinh et al., 2014; Rezende and Mohamed, 2015) allow us to construct flexible families of probability distributions on $\mathbb{R}^d$ using a change-of-variables approach. The resulting probability density function is modeled using an invertible, differential neural network $g_\phi(z)$ and a known distribution $h(\cdot)$ on $\mathbb{R}^d$, such that

$$p_\phi(x) = h\left(g_\phi^{-1}(x)\right) \left| \det \frac{\partial g_\phi^{-1}(x)}{\partial x} \right|.$$

Since g_ϕ is invertible, and assuming the Jacobian is analytically tractable, we can use the trained flow to either generate samples from p_ϕ, or to evaluate the likelihood of observations x.

When used for density estimation, normalized flows are typically trained to maximize the likelihood of the observations (possibly incorporating some regularization terms). This means we do not capture epistemic uncertainty, and are at risk of overfitting, particularly when working with small datasets. One solution might be to adopt ideas from Bayesian deep learning to approximate the Bayesian posterior, by assuming a prior distribution over the weights of the neural network (MacKay, 1992a; Graves, 2011; Neal, 1995; Blundell et al., 2015). This has two drawbacks. Firstly, posterior Bayesian inference in neural networks is typically computationally challenging. Secondly, we lose one of the key advantages of a Bayesian approach: the ability to incorporate meaningful prior knowledge. While, for example, a multivariate Gaussian prior on the weights of a neural network admits a valid posterior distribution, it is not clear how the parameters of such a prior can reasonably incorporate our prior beliefs about the data generating mechanism.

Nonparametric posterior learning (NPL, Lyddon et al., 2018) has been proposed as an alternative to classical Bayesian inference. Rather than place priors on

[2]Reproduced from the paper.

the parameter space, NPL specifies generative distributions on the space of data-generating mechanisms. In effect, we have something like a "prior" on the space of observations—a domain where we *are* likely to have prior knowledge. Inference proceeds by generating samples from this distribution over distributions, and then deterministically optimizing a function of that distribution. While conceptually straightforward and trivially parallelizable, this can be costly if each optimization is computationally demanding. For example, while NPL has been applied to small neural networks (Fong et al., 2019), this involves training a new neural network for each sample from the nonparametric posterior. This is likely infeasible in practice.

Instead, we propose an approximate NPL algorithm that partitions the normalizing flow into a shared neural network (trained once, and used for unlimited posterior samples) and sample-specific latent distributions $h_\mu(z)$ (which have an analytically tractable form and so can be easily calculated for a new posterior sample). This allows us to generate high-quality approximate nonparametric posterior samples from normalizing flows. We show that the resulting posterior distributions can achieve better generalization performance than flows trained with maximum likelihood, and that they can easily incorporate application-specific prior knowledge.

4.2 Background

4.2.1 Normalizing Flows

Normalizing flows (Tabak and Turner, 2013) are a family of generative models useful for unsupervised learning. They use an existing (potentially simple) latent distribution h and apply a change of variables to construct a new (usually more complicated) distribution p_X:

$$U \sim h \qquad\qquad x = g(u)$$

where h is the distribution of the latent variable, and g is a bijective and differentiable function that maps the latent variable $U \in \mathbb{R}^d$ to the variable of interest $X \in \mathbb{R}^d$. If

we are interested in the density of X, we can obtain it using a change of variables:

$$p_X(x) = h(g^{-1}(x)) \left| \det \frac{dg^{-1}(X)}{dX} \right|_{X=x}. \tag{4.1}$$

While some common traditional distributions fit this framework (e.g., the univariate log-normal distribution arises from taking $h = N(0,1)$ and $g(u) = \exp(\mu + \sigma u)$ with parameters μ and σ), the typical emphasis in the normalizing flow framework is constructing highly flexible transformations (or "flows") g_θ with parameters θ—usually involving the composition of many transformations—while making efforts to keep such transformations (and the Jacobian) analytically and computationally tractable. Given samples $X = \{x_1, \ldots, x_n\}$, the flow parameters θ can be estimated, for example, by maximum (log-)likelihood estimation using Equation 4.1:

$$\theta^* = \arg\max_\theta \frac{1}{n} \sum_{i=1}^{n} \log\left(h(g_\theta^{-1}(x_i))\right) + \log \left| \det \frac{dg_\theta^{-1}(X)}{dX} \right|_{X=x_i}.$$

Additionally, generating samples from the flow model is straightforward: sample from the latent distribution $U' \sim h$ and apply the flow transformation $x' = g_\theta(u')$. We can also evaluate the density of an observation x using the change-of-variables in Equation 4.1.

4.2.2 Nonparametric Learning

In the presence of only a small sample, maximum likelihood estimation will struggle to capture the uncertainty in the true data generating distribution. To better quantify that uncertainty, we can turn to a nonparametric learning approach. Nonparametric learning (NPL, Lyddon et al., 2018; Fong et al., 2019) is an inference scheme, inspired by Bayesian inference, with several key differences. Consider a sample $y_{1:n} \overset{\text{iid}}{\sim} F_0$, and a parametric family of distributions $F_\Theta = \{f_\theta(y); \theta \in \Theta\}$ indexed by parameter $\theta \in \Theta \in \mathbb{R}^p$, as in Fong et al. (2019). The Bayesian approach assumes that our true data generating distribution is contained within the family of the Bayesian posterior ($F_0 \in F_\Theta$), which may not be reasonable. NPL does not make

this assumption, instead allowing for model misspecification. In particular, NPL considers a loss function $\ell(y, \theta)$ (for example, the negative log-likelihood), seeking to estimate

$$\theta_0(F_0) = \arg\min_{\theta} \int \ell(y, \theta) dF_0(y).$$

If $\ell(y, \theta)$ is the negative log-likelihood associated with a model f_θ, θ_0 is the value of θ that minimizes $KL(f_0 \| f_\theta)$. In the Fong et al. (2019) framework, we then place a Dirichlet process prior on F_0, $F \sim \text{DP}(\alpha, F_\pi)$, where α is the concentration parameter and F_π is the base measure by which we will incorporate prior knowledge about the sampling distribution. The base measure F_π may take the form of some known density, for example, the form of the marginal likelihood of a related model $f_\pi(y) = \int f_\theta(y) d\pi(\theta)$, or may even be the empirical distribution of some historical data $\hat{y}_{1:\hat{n}}$ as $F_\pi(y) = \frac{1}{\hat{n}} \sum_{i=1}^{\hat{n}} \delta_{\hat{y}_i}(y)$. In the case of $\alpha = 0$, and a negative log-likelihood loss function, the method corresponds to the Weighted Likelihood Bootstrap (Newton and Raftery, 1994) and the Bayesian bootstrap (Rubin, 1981). Furthermore, we can obtain approximate samples from the nonparametric posterior through the posterior bootstrap by using a finite approximation to the Dirichlet process as discussed in 2.1.3 (Algorithm 2, from Fong et al. (2019)).

Algorithm 2 Posterior bootstrap sampling

Require: Observations $y_1, \ldots, y_n$, base measure F_π, number of samples B, concentration parameter α

for $b = 1, \ldots, B$ **do**

Sample m pseudo-observations $y_1^*, \ldots, y_m^* \overset{\text{iid}}{\sim} F_\pi$

Sample weights $W := (w_1, \cdots, w_n, w_1^*, \cdots, w_m^*) \sim \text{Dirichlet}\left(1, \cdots, 1, \frac{\alpha}{m}, \cdots, \frac{\alpha}{m}\right)$

$F^{(b)} = \sum_{i=1}^n w_i \delta_{y_i} + \sum_{i=1}^m w_i^* \delta_{y_i^*}$

$\theta^{(b)} = \arg\min_\theta \sum_{i=1}^n w_i \ell(y_i, \theta) + \sum_{i=1}^m w_i^* \ell(y_i^*, \theta)$

end for

return $\{\theta^{(b)}\}_{b=1}^B$

4.3 Methods

We seek to use NPL with normalizing flow models for density estimation, allowing us to incorporate prior knowledge in data space and quantify uncertainty using the NPL posterior. However, as Algorithm 2 indicates, a naïve implementation would involve performing the minimization of the normalizing flow's negative log-likelihood $\ell_{\text{NLL}}(y, \theta)$ over the flow parameters θ for each of the B samples. This is computationally intractable and in practice led to unstable estimates as each individual flow tended to overfit to a small number of (pseudo-)observations with high probability mass.

Instead, we consider a modification to NPL, where the parameter of interest can be split into a set of shared global model parameters ϕ and a set of local latent parameters θ. In particular, we focus on a setting where ϕ is computationally expensive to optimize and θ is optimized quickly given ϕ. We will jointly optimize the shared parameters ϕ by taking repeated bootstrapped samples of $F^{(b)}$ as in Algorithm 2, considering ϕ as fixed, then optimizing θ, and finally taking a gradient descent step for ϕ. As such, we require running such a process until ϕ is converged, after which the value of ϕ is fixed and we may sample the posterior of θ as before. This procedure is outlined in Algorithm 3, providing a point estimate of ϕ and approximate samples of θ.

In our normalizing flow setting, we will replace the fixed latent distribution with one parameterized by θ, and assume shared flow parameters ϕ. Our experiments use a latent Gaussian distribution with mean μ and identity covariance, such that $\theta = \mu$. In this case, the maximum likelihood estimate required to sample $\theta^{(b)}$ is simply given by the weighted mean of the preimage of the augmented observations:

$$\mu^{(b)} = \sum_{i=1}^{n} w_i g_{\hat{\phi}}^{-1}(y_i) + \sum_{i=1}^{m} w_i^* g_{\hat{\phi}}^{-1}(y_i^*).$$

For a latent normal distribution with covariance matrix Σ, we can similarly arrive at a simple closed-form expression, although in practice this seemed to yield numerical

Algorithm 3 Posterior Bootstrap with Shared Parameters

Require: Observations $y_1, \ldots, y_n$, base measure F_π, initial shared parameter value ϕ_0, number of samples B, concentration parameter α, learning rate τ

$\phi \leftarrow \phi_0$

while not converged **do**

Sample m pseudo-observations $y_1^*, \ldots, y_m^* \sim F_\pi$

Sample weights $W := (w_1, \cdots, w_n, w_1^*, \cdots, w_m^*) \sim \text{Dir}\left(1, \cdots, 1, \frac{\alpha}{m}, \cdots, \frac{\alpha}{m}\right)$

$\tilde{F} = \sum_{i=1}^n w_i \delta_{y_i} + \sum_{i=1}^m w_i^* \delta_{y_i^*}$

$\theta = \arg\min_\theta \int \ell(y, \phi, \theta) d\tilde{F}(y)$

$\phi \leftarrow \phi + \tau \nabla \left(\sum_{i=1}^n w_i \ell(y_i, \phi, \theta) + \sum_{i=1}^m w_i^* \ell(y_i^*, \phi, \theta) \right)$

end while

$\hat{\phi} \leftarrow \phi$

for $b = 1, \ldots, B$ **do**

Sample m pseudo-observations $y_1^*, \ldots, y_m^* \sim F_\pi$

Sample weights $W := (w_1, \cdots, w_n, w_1^*, \cdots, w_m^*) \sim \text{Dir}\left(1, \cdots, 1, \frac{\alpha}{m}, \cdots, \frac{\alpha}{m}\right)$

$F^{(b)} = \sum_{i=1}^n w_i \delta_{y_i} + \sum_{i=1}^m w_i^* \delta_{y_i^*}$

$\theta^{(b)} = \arg\min_\theta \int \ell(y, \hat{\phi}, \theta) dF^{(b)}(y)$

end for

return $\{\theta^{(b)}\}_{b=1}^B, \hat{\phi}$

instabilities due to outliers with large weights distorting the estimates of the latent distribution parameters.

This augmented normalizing flow model dovetails with Algorithm 3. The optimization step required for each bootstrap sample is reduced to a closed-form function of the preimage. Furthermore, additional samples from the nonparametric posterior can be obtained in a straightforward way, without the computational cost of learning additional flows. We compare the performance of our approach with the naïve NPL implementation in §4.5.4.

4.4 Related Work[3]

This paper considers the nonparametric posterior distribution over a normalizing flow as a way of incorporating prior information and capturing epistemic uncertainty. In the broader context of neural networks, a number of alternative approaches have been proposed to either capture epistemic uncertainty, or to incorporate prior information; however these approaches have mostly not been used with normalizing flows.

Several works have applied a classical Bayesian approach to learning neural networks, placing priors on the weights and updating them using Laplace approximations (MacKay, 1992a; Daxberger et al., 2021), variational inference (Graves, 2011; Blundell et al., 2015), or assumed density filtering (Hernández-Lobato and Adams, 2015). Of particular relevance to this paper, Trippe and Turner (2018) use a variational inference approach to learn conditional normalizing flows. However, these Bayesian methods tend to be slower than non-Bayesian approaches and do not allow us to easily incorporate domain knowledge. Gal and Ghahramani (2016) showed that Monte Carlo dropout can be interpreted as approximate Bayesian inference in a neural network, providing a lightweight alternative to the aforementioned Bayesian approaches; however, again, we cannot incorporate domain-specific knowledge and have minimal ability to adjust the priors on the weights.

Other works have employed bootstrapping approaches to capture uncertainty in neural networks. Franke and Neumann (2000) show consistency of the frequentist bootstrap for neural networks. Schulam and Saria (2019) use influence functions to estimate how the predictions of a neural network would have changed if it were trained on a bootstrapped version of the original dataset; however, this is computationally costly due to the need to compute the Hessian matrix. In an attempt to reduce the cost of frequentist bootstrap approaches, several papers have aimed to approximate

[3]Reproduced from the paper.

the bootstrapping procedure. In the context of reinforcement learning, Osband et al. (2016) reduces the computational complexity of bootstrapping by partitioning the architecture into a shared neural network (updated by all bootstrapped samples), and local "heads" that branch off this shared neural network and are only updated based on a single bootstrap sample. Based on a similar partition, Shin et al. (2021) multiplicatively incorporates bootstrap weights in the final layer of a neural network, learning a shared base network that can be combined with new weights. These partitioning approaches have some of the flavor of the partitioning used in this paper, however, the architectures they assume do not naturally translate to normalizing flows.

Less attention has been given to incorporating meaningful prior information while training neural networks. In Osband et al. (2018), the authors show that, in the case of a linear model, we can sample from the Bayesian posterior by solving an optimization problem using perturbed data samples, and a random regularization term. While the direct link to a Bayesian posterior does not hold in the non-linear case, they make use of this intuition to incorporate the idea to obtain approximate posterior samples in a neural-network-based reinforcement learning model. The observations are perturbed—either by adding Gaussian noise, or by taking a bootstrap sample—and a sample from some prior distributions over functions is incorporated as a regularizer. While the mechanism for incorporating a prior is different from that proposed in this paper, it shares the fact that the "prior" is specified in the function space, rather than the parameter space.

4.5 Evaluation

We have developed a framework to incorporate prior information in a normalizing flow model, which we expect can improve posterior uncertainty estimates, particularly in the small data regime. However, our approximate NPL method introduced an additional optimization step for jointly learned shared parameters, which

could (in principle) inhibit the learning of the flow's parameters as compared to the naïve NPL or maximum likelihood approaches. We investigate our proposed method first with a qualitative exploration of the behavior in a simple simulation study. Then, we apply our method to a set of datasets following Papamakarios et al. (2017). Finally, we present a short comparison between our method using shared flow parameters and the naïve NPL implementation.

4.5.1 Implementation Details

Using what we learned from some initial exploration of the Gas dataset (see §B), we determined some hyperparameters related to network depth and structure to best explore how our model compares against a baseline flow. We also considered an initialization scheme where the flow was first trained to the identity function to prevent numerical issues but abandoned this effort as it seemed not to improve performance. Additionally, we discovered that a full-covariance latent distribution tended to be numerically unstable, so we focused on a latent Gaussian distribution with identity covariance. More details available in §B.

In particular, the following experiments incorporated:

- A 3 block masked autoregressive flow (Papamakarios et al., 2017), including batch normalization layers, with a multivariate normal base distribution.
- Five random initializations, selecting the best model based on validation set average log-likelihood.
- For convergence, we applied early stopping when validation set average log-likelihood did not improve for 30 epochs (maximum number of epochs in practice was 5478).
- A standard normal distribution as the base distribution for the baseline comparison, and a multivariate normal with learned mean for the WLB and NPL runs.

- A subset of the training data ($N \in \{500, 1000, 5000, 10000\}$) to explore the small data regime where we would expect the robustness of the WLB or NPL approaches to assist the model.

- Two priors with concentration $\alpha = 100$ for the NPL approach. One used the mean and covariance of the training data subset to inform a multivariate normal prior distribution. One used additional training data (outside the selected subset) as pseudo-observations, essentially mimicking using historical data in a best-case scenario.

- We report the average log-likelihood of the test set, $\overline{\ell_{\text{test}}}$. In the WLB and NPL cases, we use 100 samples of Dirichlet weights and pseudo-observations to determine the base distribution and report the average $\overline{\ell_{\text{test}}}$ from those 100 samples.

4.5.2 Qualitative Evaluation on Synthetic Data

To explore the performance and behavior of our approach, we first consider an example of two-dimensional synthetic data. We applied our method to a simple Gaussian mixture model with two equal-weight components in two dimensions, seeking to explore how different base distributions F_π and concentration parameters α impact the nonparametric posterior (see Figure 4.1). We expect that a larger concentration α will cause the posterior to be pulled toward F_π. And indeed, for $\alpha = 0$, we observe that the samples remain similar to those of the original flow trained with maximum-likelihood estimation. At $\alpha = 1$, we observe subtle differences, and at $\alpha = 100$ (effectively giving the prior equal weight to the 100 observations), we see the posterior samples clearly reflecting the prior. Additionally, we clearly observe the effect of different base distributions F_π at this high concentration, with the samples from the model with the higher-variance prior being more spread out.

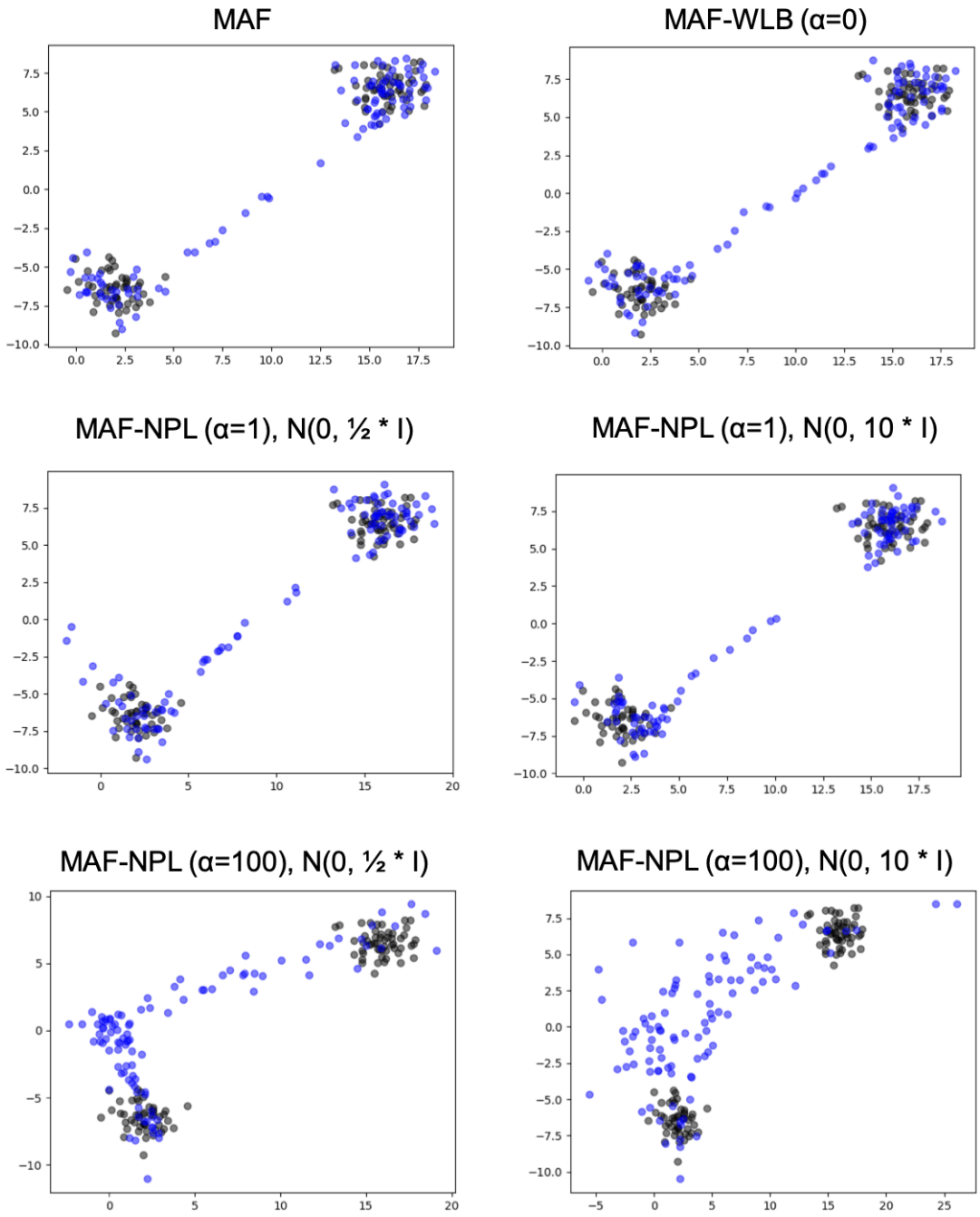

Figure 4.1: Samples (blue) from MAF model trained on $N = 100$ data points (black), in the original MAF model, and in the posterior normalizing flow version with $\alpha \in \{0, 1, 100\}$ and two spherical Gaussian priors centered at $(0, 0)^\top$.

4.5.3 Quantitative Experiments on Real Datasets

We sought to determine how our method compares against a maximum likelihood approach by varying the size of the training subset. Additionally, for our method, we explored a version that did not incorporate pseudo-observations ($\alpha = 0$, corresponding to a weighted likelihood bootstrap version of our method), and compared two using two "prior" base distributions: an empirical distribution derived from additional observations from outside the training subset (effectively giving our model more training data, representing a best-case scenario) and a multivariate Gaussian with mean and covariance determined by the training subset.

Following Papamakarios et al. (2017), we present results on several UCI datasets, along with the BSDS300 dataset, with sizes shown in Table 4.1. Our test set log-likelihood results are shown in Figure 4.2.

	BSDS300	Gas	HEPMASS	MiniBooNE	Power
N_{train}	1000000	852174	315123	29556	1659917
N_{val}	50000	94685	35013	3284	184435
N_{test}	250000	105206	174987	3648	204928
D	63	8	21	43	6

Table 4.1: Properties of datasets used in experiments, following Papamakarios et al. (2017).

Overall, our method tends to attain a higher average test-set log-likelihood than the maximum-likelihood approach when fewer observations are used in training. In particular, even using a Gaussian base distribution often appears helpful, indicating that our method is able to reduce overfitting, even when essentially no new information is provided. We do observe that the $\alpha = 0$ case does tend to result in instabilities. We suspect this is due to its tendency to overfit to a small number of observations with high probability mass due to the Dirichlet weights. We do tend to observe that as N increases in each dataset, all four methods tend to improve in overall test set log-likelihood (with some exceptions). Furthermore, we do

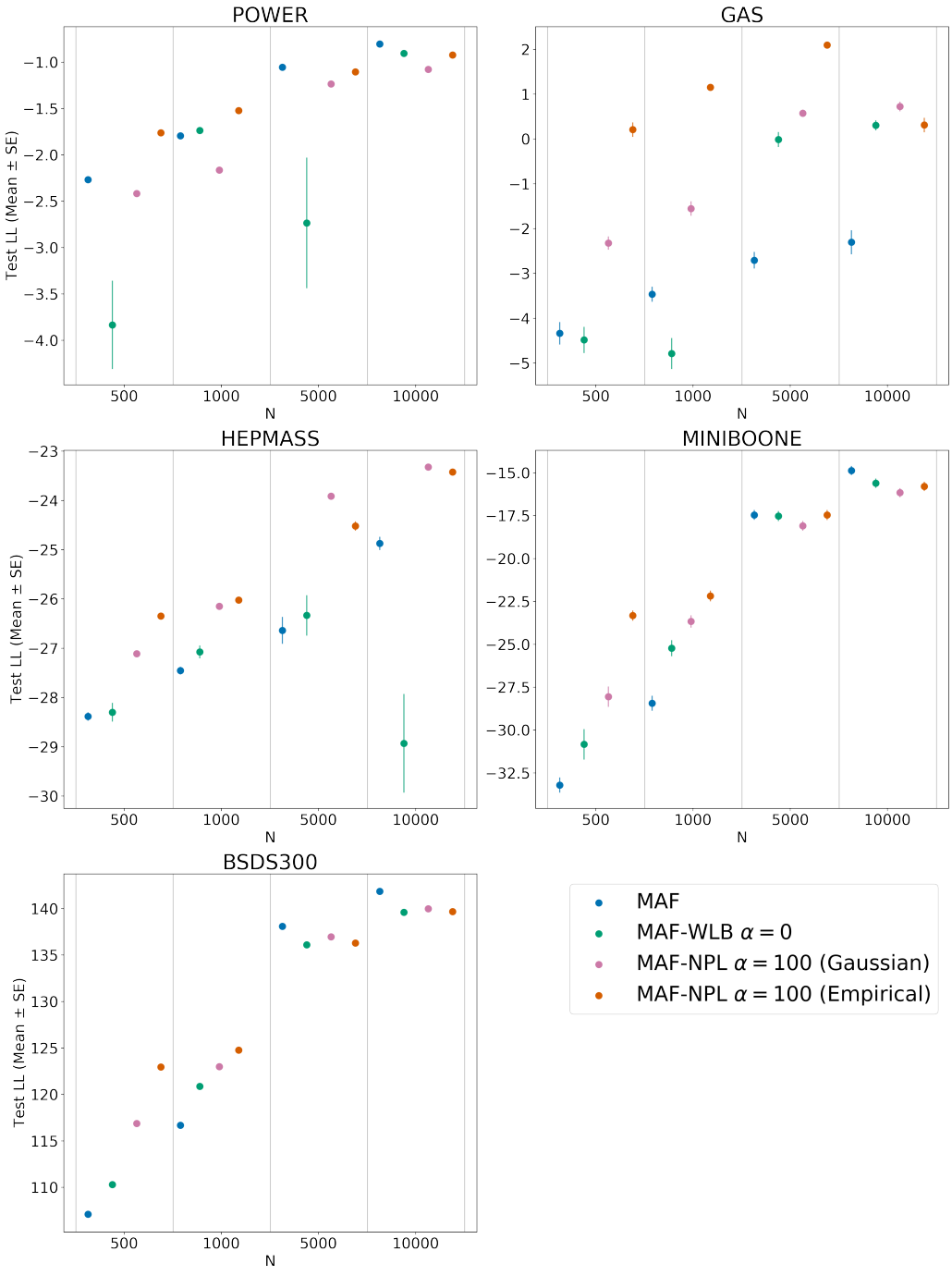

Figure 4.2: Results on datasets, with covariate dimension increasing

observe that for larger training subsets $N \in \{5000, 10000\}$, we occasionally see our method underperforming the baseline. We suspect that for some datasets, the effect of the Dirichlet weights will under-weight enough training datapoints that the NPL-based methods are less able to capture aleatoric (data) uncertainty compared to the maximum-likelihood approach.

4.5.4 Comparison with Naïve NPL Approach

As discussed in §4.3, a naïve nonparametric learning approach to normalizing flows would learn all model parameters locally (rather than splitting into the jointly-learned flow parameters ϕ and local latent distribution parameters θ), which, as discussed, will usually be computationally prohibitive. However, as an exploration, we applied this method to the MiniBooNE dataset with $N = 1000$ training examples. We learned 100 models under the NPL framework with $\alpha = 0$ (corresponding to the WLB) and a standard normal base distribution, each of which was the best-of-five as judged by validation set average log-likelihood, to try to compare fairly to the best-of-five models learned using our method (which we employed to account for numerical instabilities).

Results are in Table 4.2. The maximum likelihood and naïve WLB implementations both have a standard Gaussian latent distribution. The three versions of our method learn the mean of a multivariate Gaussian latent distribution ($\theta = \{\mu\}$). In this limited exploration, we observe an improved test log-likelihood when using our method, likely as a result of learning the flow parameters globally, which acts as a sort of regularization in the WLB setting compared to the naïve implementation or the maximum likelihood method. In fact, not only does the empirical base distribution result in a better test log-likelihood (as expected for what amounts to an additional set of training points), but even the Gaussian base distribution based on the training subset improves the test log-likelihood, indicating that the additional less-informative pseudo-observations still help any model instabilities.

Method	Average Test LL
Maximum Likelihood, $\mu = 0$	-28.441 ± 0.405
Naïve WLB, $\mu = 0$	-31.975 ± 0.453
Ours, WLB, $\theta = \mu$	-25.234 ± 0.438
Ours, NPL $\alpha = 100$ (Gaussian), $\theta = \mu$	-23.671 ± 0.327
Ours, NPL $\alpha = 100$ (Empirical), $\theta = \mu$	-22.188 ± 0.278

Table 4.2: Comparison of standard NPL approach to our version on the MiniBooNE dataset ($N = 1000$).

4.6 Discussion and Future Work

The masked autoregressive flow (MAF, Papamakarios et al., 2017) may not be the ideal choice of flow for these datasets, so one potential area of exploration is using other flow models, particularly those more relevant to particular datasets. MAF is ideal for datasets where each example's covariates relate to one another in an autoregressive way (e.g., time series) and the UCI datasets investigated (Gas, HEPMASS, MiniBooNE, Power) do not seem to be structured in this way (although in some cases, each example is itself part of a time series but treated as i.i.d.). Examples of the BSDS300 are derived from natural image patches, which may benefit from the autoregressive treatment in MAF. Other choices of flows are explored further as joint work in Ott and Williamson (2022a).

Here, we presented two plausible priors, but we believe the nonparametric posterior normalizing flow would be best suited to a problem with historical data or a related dataset. In particular, this does present an interesting framework for an analogue of inductive transfer learning, using the empirical distribution of e.g., a large heterogeneous dataset as a prior, with a smaller homogenous dataset being the dataset of interest. We hope to explore such problems in future work.

Chapter 5: Edge-Based Generative Graph Neural Networks[1]

Abstract

Graph neural networks (GNNs, Scarselli et al., 2008) are neural network-based models for graph-structured data. Recent work in generative GNN models allows for generating novel graphs similar to those from a given data-generating distribution. However, existing approaches to GNNs tend to generate graph sequences that are dense, while many real-world graph sequences are sparse. Building on recent work on edge-exchangeable graph models (Crane and Dempsey, 2016; Cai et al., 2016) that produce sparse graph sequences, we propose an edge-based generative GNN model for sparse graphs.

5.1 Introduction

Graphs are structures that express relational information. Many types of data can be expressed using graphs, like molecules, social networks, or email interactions. Sampling graphs from a distribution can be useful in problems like drug discovery, where molecules are modeled using graphs and we seek to sample novel molecules (Gómez-Bombarelli et al., 2018; Li et al., 2018b). Similarly, problems in neural architecture search (Xie et al., 2019) or in network science generally may find use of generating novel graphs from a distribution, potentially learned from graphs in a

[1]This presents work that is part of an ongoing collaboration myself, Curtis Carter, Elahe Ghalebi, and Sinead Williamson. The work contained in this chapter represents my contribution to this collaboration.

particular domain.

Graph neural networks (GNNs Scarselli et al., 2008) and graph convolutional networks (GCNs Welling and Kipf, 2016) are neural network-based approaches to modeling graphs, creating latent representations of nodes known as node "embeddings." These node embeddings provide a graph-informed representation of the node useful for a variety of problems, including node classification, graph classification, or link prediction. Recent work on generative GNN models extends the GNN paradigm to learn a distribution over graphs, generating graphs from that distribution (Simonovsky and Komodakis, 2018; You et al., 2018; Liao et al., 2019; Li et al., 2018b).

Whereas most generative GNN models have not focused on the theoretical properties (e.g., sparsity, degree distribution, clustering coefficients) of the distributions they learn, the properties of non-GNN-based probabilistic models of graphs are an active and developed area of research. For example, random vertex-exchangeable simple graphs (that is, undirected, without self-loops) can be parameterized by a random graphon, which are provably dense (or empty) almost surely (Orbanz and Roy, 2014). Kronecker graphs generate graphs that are self-similar. Barabási-Albert (BA, Barabási and Albert, 1999) graphs have a power law degree distribution. Sparse exchangeable graphs based on Poisson processes (Caron and Fox, 2017) and and edge-exchangeable graphs based on nonparametric distributions (Crane and Dempsey, 2016; Cai et al., 2016) generate sparse graph sequences.

Departing from existing generative GNN models that generate graphs in a manner similar to vertex-exchangeable methods, we explore how edge-exchangeable graphs can be incorporated in a generative GNN setting, allowing us to capture graph sparsity, which we observe in many real-world graphs. We begin with further background on probabilistic graph models and generative GNN models, then present our method and experiments, including qualitative and quantitative results.

5.2 Background

We begin by reviewing existing models for random graphs and graph neural networks.

5.2.1 Models for Random Graphs

Graphs from real-world problems can be described by a variety of graph statistics. Some of these include graph's sparsity, degree distribution, local clustering coefficient distribution, and Laplacian spectrum distribution (see §C.3.1 for a brief introduction to these statistics). As such, various models for graphs have been proposed, attempting to replicate observed behavior. Of these, many interesting models are vertex-exchangeable models.

Vertex-exchangeable models (Aldous, 1981; Hoover, 1979) of graphs assign equal probability to two graphs if they are the same up to a permutation of the vertices. Said another way, the model is invariant to jointly permuting the rows and columns of a graph's adjacency matrix. The Erdős-Renyi model (Erdős et al., 1960) is vertex-exchangeable: given a number of nodes n and edge probability p, each possible edge in the network is realized with probability p. The stochastic block model (Holland et al., 1983) expands on this idea, with each node being assigned to a cluster, and edge probabilities within- and between-clusters as model parameters.

While every graph has a particular density ($|E|/(|V|(|V|-1))$ for simple graphs), if we have a sequence of graphs $(G_n)_n$, we can consider whether the graph sequence is dense or sparse as $n \to \infty$, by comparing the rates of growth of $|V_n|$ and $|E_n|$, the number of nodes and number of edges in graph G_n. A sequence is dense if $|E_n| = \Omega(|V_n|^2)$ and sparse if $|E_n| = o(|V_n|^2)$ (Cai et al., 2016). Existing work has shown that many real-world graphs tend to grow in a sparse way. For example, in many social networks, the number of edges associated with a new user is subquadratic in the number of users—most users are not connected to most users. However, under the Aldous-Hoover theorem (Aldous, 1981; Hoover, 1979), it can be shown that

vertex-exchangeable approaches yield dense (or empty) graph sequences. Thus, graph models that use a vertex-exchangeable approach are misspecified for many real-world graph problems.

Instead, we can consider models that produce sparse graphs. For example, Barabási-Albert (BA, Barabási and Albert, 1999) graphs are preferential attachment models that grow the graph by adding each new node along with connections to m existing nodes, with the likelihood of attachment being proportional to the degree of the existing node. As a result, BA graph sequences are sparse, with a power-law degree distribution. More relevant to our approach are edge-exchangeable graphs (Crane and Dempsey, 2016; Cai et al., 2016), which consider graphs to have equal probability if they are the same up to a permutation of the edges. Edge-exchangeable graphs have been shown to produce sparse graph sequences (under some conditions, including having a sufficiently large probability of selecting a new node as the number of edges grows). Furthermore, of interest are other edge-based approaches to graphs, inspired by edge-exchangeable models, like the dynamic nonparametric network distribution (DNND, Ghalebi et al., 2019), that are able to capture additional structure and dynamics of graph sequences, while still producing sparse graph sequences.

5.2.2 Graph Neural Networks

Graph neural networks (GNNs, Scarselli et al., 2008) are a neural network-based approach to problems involving graph-structured data. As discussed in §2.2.1, they often involve constructing an embedding for each node, often involving a message-passing scheme. The original GNN approach develops node embeddings x_v by constructing messages (combining any node or edge covariates ℓ_u and $\ell_{(v,u)}$ and any current node embeddings) from node u to node v using a learnable function h_w, aggregating them over the neighborhood of node v:

$$x_v = \sum_{u \in \mathcal{N}(v)} h_w(\ell_v, \ell_{(v,u)}, x_u, \ell_u), \quad v \in V.$$

This procedure can be iterated (to a fixed-point solution in the construction above) with the final node embeddings used in problems such as node classification or link prediction, or aggregated to form a graph embedding for problems like graph classification.

The original GNN approach has been extended in many ways, including generative GNN models (e.g., Kipf and Welling (2016); Li et al. (2018b); You et al. (2018); Grover et al. (2019); Liao et al. (2019)). We review in detail the GraphRNN (You et al., 2018) and DeepGMG (Li et al., 2018b) models in §2.2.1.1. DeepGMG constructs simple graphs in a node-based way, similar to the process used in preferential attachment graphs like BA graphs. DeepGMG determines whether to add a new node based on the current graph embedding (an aggregated projection of the current node embeddings). If a new node v_* is added to the graph, edges are sampled between v_* and existing nodes based on the node embeddings. As a generative model, DeepGMG is able to learn a process to generate new graphs similar to those in an existing dataset.

GraphRNN (You et al., 2018) also formulates the graph generation process in a node-based way for simple graphs. GraphRNN considers the sequence of adjacency vectors that comprise the adjacency matrix and maintains a graph embedding using an RNN (for a brief introduction on RNNs, see §C.5). As each new node is added to the graph, its edges to existing nodes are added probabilistically based on the graph embedding (either independently or dependently using an RNN by considering each existing node in turn). Again, novel graphs can be generated that are similar to those of an existing dataset.

Variational graph auto-encoders (VGAEs, Kipf and Welling, 2016) are an alternate method of using GNN-inspired models in a generative setting. VGAE uses a graph convolutional network as an encoder to construct the mean and variance of a per-node variational Gaussian latent distribution. Samples from the latent distribution are then applied to a simple inner product model (that could in principle be

replaced with a neural network) that provides a distribution on adjacency matrices. From this construction, VGAE can be trained on a graph, yielding the variational posterior latent distribution, which can then be used to sample missing edges of the graph or potentially to generate novel but similar graphs.

In general, the existing generative GNN models essentially focus on sampling whether an edge is present or absent, constructing the adjacency matrix. This construction is similar to graphon-based methods (which correspond to vertex-exchangeable graph sequences), where elements of the adjacency matrix are sampled from some distribution. While the theoretical properties of the distributions induced on graphs by these models have not been thoroughly explored, we do know that vertex-exchangeable models cannot produce sparse graph sequences. Instead, we seek to explore an edge-based generative GNN model, expecting that we will better be able to model sparse graphs, which are relevant to many real-world graphs.

5.3 Method

Approaches like edge-exchangeable graph sequences are provably capable of generating sparse graphs. While generative graph neural network-based approaches offer far more flexibility in modelling, they have not focused on producing graphs that are sparse. We aim to combine these two areas of work, creating a GNN-based generative model that can generate sparse graphs. In particular, we seek to combine the structure of representation used in generative GNNs like DeepGMG (Li et al., 2018b) with the edge-based methods capable of producing sparse graphs such as Cai et al. (2016); Crane and Dempsey (2016); Ghalebi et al. (2019).

Much like edge-exchangeable models discussed in §5.2, our method generates graphs edge-by-edge. We will form a distribution over graphs constructed from a sequence of (an unbounded number of) edges. As such, graphs generated by our model will be able to grow over time, in a manner similar to many real-world settings such as social media or interaction graphs. Of course, similar to edge-exchangeable

approaches, we can specify a finite number of edges to consider and construct the relevant graph.

Our method constructs and iteratively updates embeddings for each node, which are projected and aggregated into an overall graph embedding, using these embeddings to inform the choice of the nodes that appear in the next edge. While we will treat edges as undirected for the purpose of message passing, we will sample the edges (u, v) as ordered pairs, calling node u the "sender" and v the "recipient." We sample the nodes in each edge sequentially, selecting the node according to a $K + 1$-dimensional probability vector, where K is the number of previously seen nodes (including the sender if we are selecting the recipient). As an important note, graphs constructed with our method need not be simple; repeated edges and self-loops are permissible in our model.

The probability of selecting a particular node depends on the node (and graph) embeddings. When sampling the sender for the new edge (based on the current state of the graph), this takes the form

$$p(S = v) \propto \begin{cases} f_{\text{sender}}(h_v; \theta) & v \in V_n, \\ f_{\text{new sender}}(h_G; \theta) & v = |V_n| + 1, \end{cases} \tag{5.1}$$

where V_n are the nodes present in the graph after n edges, h_v is the embedding of node v, h_G is the graph embedding (a gated sum of projections of node embeddings, following Li et al. (2018b)), neural networks f_{sender} and $f_{\text{new sender}}$, which depend on learnable parameters θ (where we have simplified notation, treating θ as the set of all parameters learned in the model). Similarly, when selecting the recipient, we have

$$p(R = v | S = s) \propto \begin{cases} f_{\text{recipient}}(h_v, h_s; \theta) & v \in V_n \cup \{s\}, \\ f_{\text{new recipient}}(h_G, h_s; \theta) & v = |V_n \cup \{s\}| + 1. \end{cases} \tag{5.2}$$

When sampling either the sender or recipient, if a new node is selected, we initialize its embedding based on the current graph state: $h_v = f_{\text{init}}(h_G, \theta)$[2].

[2]We also explored initial embeddings including $h_v \sim U([0, 1]^d)$ but found that the version based on the current graph embedding worked best.

A visual representation of this edge-based graph generation process is in Figure 5.1, where nodes and edges are both labeled and edges. In each of the 16 steps shown, a new edge is added, which may or may not add a new node (or two new nodes) to the graph.

After sampling a new edge (s, r), we update the node embeddings using message passing. First, we construct messages from all nodes u to their neighboring nodes $v \in \mathcal{N}(u)$ based on the current node embeddings:

$$m_{v \leftarrow u} = f_{\text{message}}(h_v, h_u; \theta). \tag{5.3}$$

We will then update the node embeddings using an RNN that depends on the current node embedding and an aggregation of the incoming messages m_v to node v:

$$h_v = f_{\text{update}}(h_v, m_v; \theta). \tag{5.4}$$

When aggregating these messages, we explored two variations. For the first option, we simply add the incoming messages over all edges in the current graph, treating edges as undirected, but with repeated edges and self-loops allowed:

$$m_v^{\text{no time}} = \sum_{(u,v) \in E_{\text{undirected}}} m_{v \leftarrow u}. \tag{5.5}$$

Secondly, we explored a time-aware variation. Here, every edge is additionally given a timestamp (which is non-decreasing as we add additional edges), and we weight messages sent over each edge by a function of how much time has elapsed since that edge was added to the graph. After adding an edge at time t, we construct the weights for each edge e that was added at time t_e and aggregate the messages as:

$$m_v^{\text{time}}(t) = \sum_{u \in \mathcal{N}(v)} \left(\sum_{e=(u,v) \in E_{\text{undirected}}} f_{\text{time}}\left(\frac{1}{t - t_e + 1}; \theta \right) \right) m_{v \leftarrow u}. \tag{5.6}$$

As such, we expect this time-aware version to more readily capture any temporal dynamics in a particular graph distribution. The full learning algorithm is given in Algorithm 4.

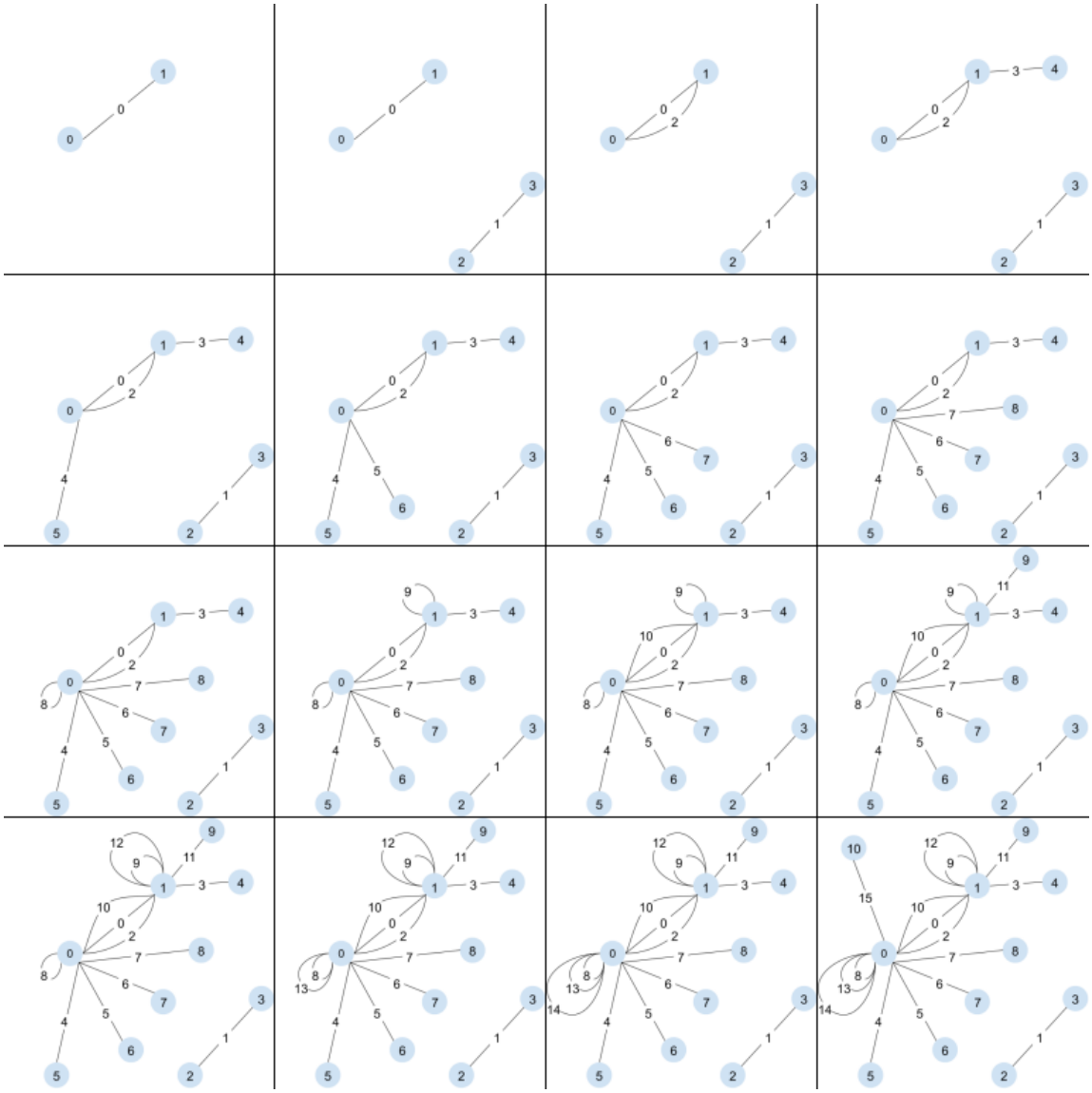

Figure 5.1: Demonstration of how the graph is generated in an edge-based method like ours or the BHP, with nodes and edges labeled.

Algorithm 4 Edge-based generative graph neural network

Require: Graph edge lists $\mathcal{E} = [E^{(1)}, \ldots, E^{(N)}]$, graph edge timestamps $\mathcal{T} = [T^{(1)}, \ldots, T^{(N)}]$
  while not converged **do**
    for $g = 1, \ldots, N$ **do**
      Initialize graph $G = \{V =, E =\}$
      for $t \in T^{(g)}$ **do**
        Select sender s according to Equation 5.1.
        If the sender is a new node ($s \notin V$), initialize $h_s = f_{\text{init}}(h_G, \theta)$.
        Select sender r according to Equation 5.2.
        If recipient is a new node ($r \notin V \cup \{s\}$, initialize $h_r = f_{\text{init}}(h_G, \theta)$.
        Add edge (s, r) to E at time t (and nodes s, r to V as appropriate).
        Construct messages $m_{v \leftarrow u}$ according to Equation 5.3.
        Aggregate messages to form m_v according to Equation 5.5 or 5.6.
        Update node embeddings h_v for all nodes according to Equation 5.4.
      end for
      $\theta \leftarrow \theta + \nabla \ell(E^{(g)}; \theta)$.
    end for
  end while

5.3.1 Generating Sparse Graph Sequences

We now consider how our edge-based model can recover graph sparsity, which is relevant in many real-world graph settings. We begin by considering the behavior of edge-exchangeable graphs. Edge-exchangeable graphs construct a sequence of edges by repeatedly sampling pairs of vertices from a distribution on the space of edges. Typically, this distribution is of the form $W \times W$, where W is sampled from a nonparametric distribution. If this distribution has heavy tails – such that the probability of an edge introducing a previously-unseen vertex is sufficiently large – the resulting graph is sparse. In particular, a normalized generalized gamma process (Cai et al., 2016) or a Pitman-Yor process (Crane and Dempsey, 2016) is capable of generating sparse graphs (see (Cai et al., 2016) for more details on the technical conditions).

Our graph generative model constructs distributions over edges, where there is always a finite probability of seeing new nodes. This implies that the model can

capture the behavior of a (dynamically evolving) nonparametric distribution, where the number of nodes is unbounded as the graph grows. While this distribution is not inherently heavy-tailed, it can capture heavy-tailed behavior.

As a simple example of capturing heavy-tailed behavior, consider the "binary Hollywood process (BHP)," a Pitman-Yor-based graph of Crane and Dempsey (2016). The BHP produces edge-exchangeable graph sequences (Crane and Dempsey, 2016, Theorem 5.1) with preferential attachment. BHP samples edges E_n, first selecting a sender $E_{n,1}$, then the recipient $E_{n,2}$, creating a graph $G_n = G_{n-1} \cup E_n$. Based on the most recent graph G_{n-1}, BHP selects the new node based on:

$$P(E_{n,j} = i | E_{<n,1:2}, E_{n,<j}) \propto \begin{cases} D_{n,j}(i) - \sigma, & i = 1, \ldots, N_{n,j} \\ \alpha + \sigma N_{n,j}, & i = N_{n,j} + 1 \end{cases}, \quad j = 1, 2, \tag{5.7}$$
$$N_{n,j} = \left| V_n \cup \{E_{n,k}\}_{k=1}^{j-1} \right|$$

where $N_{n,j}$ is the number of nodes in the current graph (including the sender $E_{n,1}$ if $j = 2$) and $D_{n,j}(i)$ is the number of times node i has been seen (again, counting the sender if $j = 2$). In the case of $0 < \sigma < 1$ and $\alpha > -\sigma$, which represents an infinite population of available nodes (such that $\lim_{n\to\infty} |V_n| \to \infty$ almost surely), the graph sequence will be sparse (whether sparsity is measured for the multigraph or the graph projected to exclude repeated edges, see Crane and Dempsey (2016), Theorem 5.3 and 5.4).

Our model can recover this distribution if the node embeddings capture the degree of that node (minus a constant), and the overall graph embedding captures the number of observed vertices (minus a constant). Both of these embeddings are achievable via a GNN, implying that our model has the ability to model graph sparsity. More generally, we expect to have sparse graphs provided the probability of seeing a new node is at least $\frac{\alpha + |V_n|\sigma}{2n + \alpha}$ for some $0 < \sigma \leq 1$.

5.4 Evaluation

In this work, we focus on the task of generating graphs similar to those in a dataset of interest. First, we demonstrate the ability of our model to capture graph sparsity. Next, we compare our model with deep generative models of graphs (DeepGMG, Li et al., 2018b), a node-based generative GNN model, by assessing performance on a number of synthetic and real training datasets, described below.

5.4.1 Qualitative Exploration

As discussed in §5.3.1, we discussed how our approach has the capacity to model sparse graphs, using the Pitman-Yor-based binary Hollywood process (BHP) graphs of Crane and Dempsey (2016) as an example. Here, we demonstrate this capacity empirically. We trained our model on 500 BHP graphs with number of edges selected uniformly at random from 50 to 500, using an additional 100 graphs for validation and 100 for testing (here, used only for visualization). We used BHP parameters $\alpha = 1$ and $\sigma = 0.7$, which represent a distribution over sparse graphs with an infinite population of available nodes. Figure 5.2 includes an example BHP graph from the test set.

First, we consider the probability distribution for the ith sender, conditioned on the previous $i-1$ edges of the graph. Figure 5.2 shows the true probabilities under the BHP model and the probabilities recovered under our model. We observe that our model is able to capture the behavior of the generating distribution.

Next, we explore whether we recover sparsity when sampling novel graphs, using the model trained on BHP graphs. Figure 5.3 show the number of nodes in graphs generated using the BHP and graphs generated using our model as we increase the number of edges. We observe that graphs generated by our model are in fact slightly sparser than those generated by the BHP (with more nodes for a fixed number of edges), however the two distributions overlap, and remain close to the

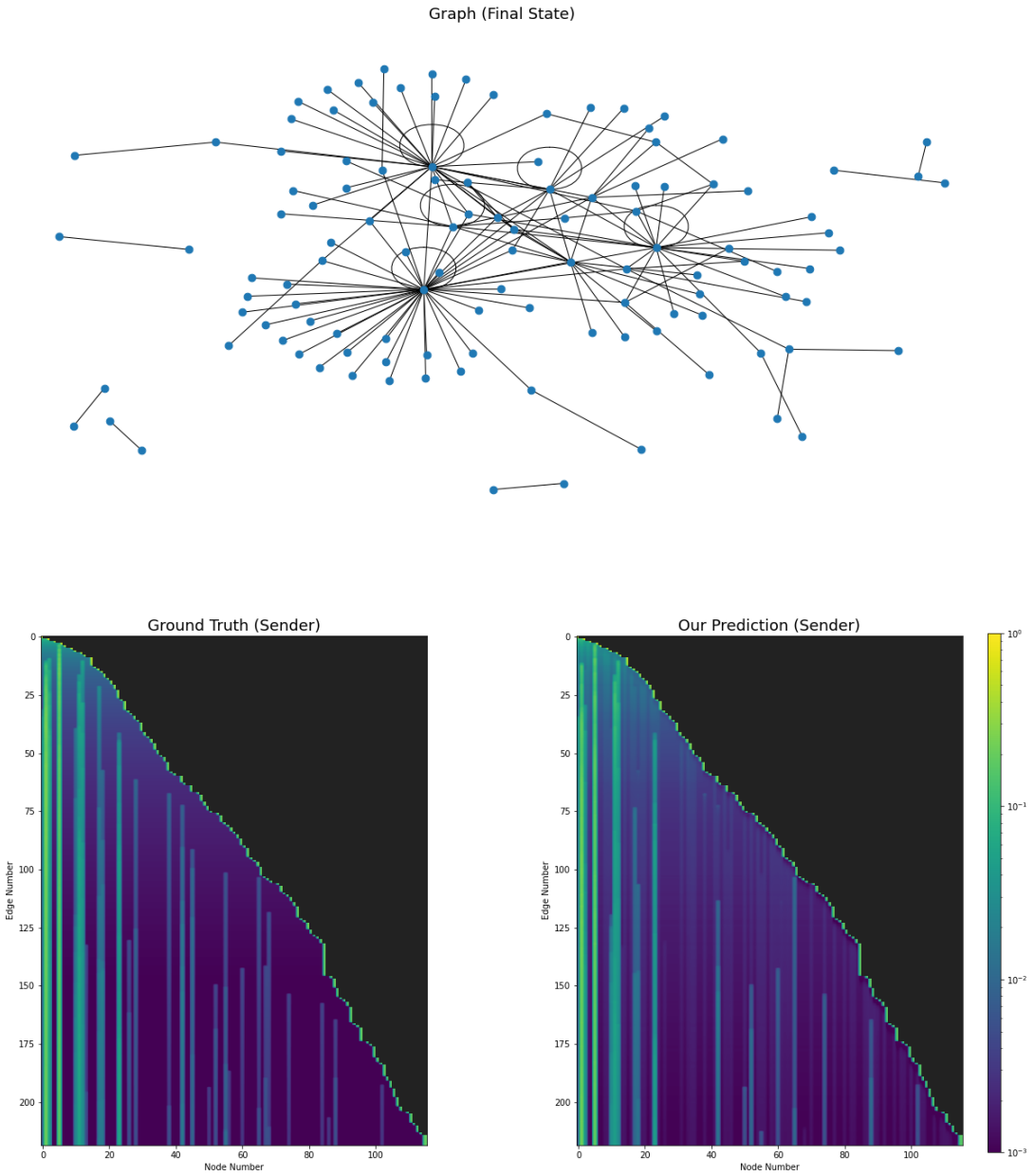

Figure 5.2: *Top*, a BHP graph from the test set. *Left*, ground truth sender probabilities under the BHP model. *Right*, Predicted sender probabilities within our model.

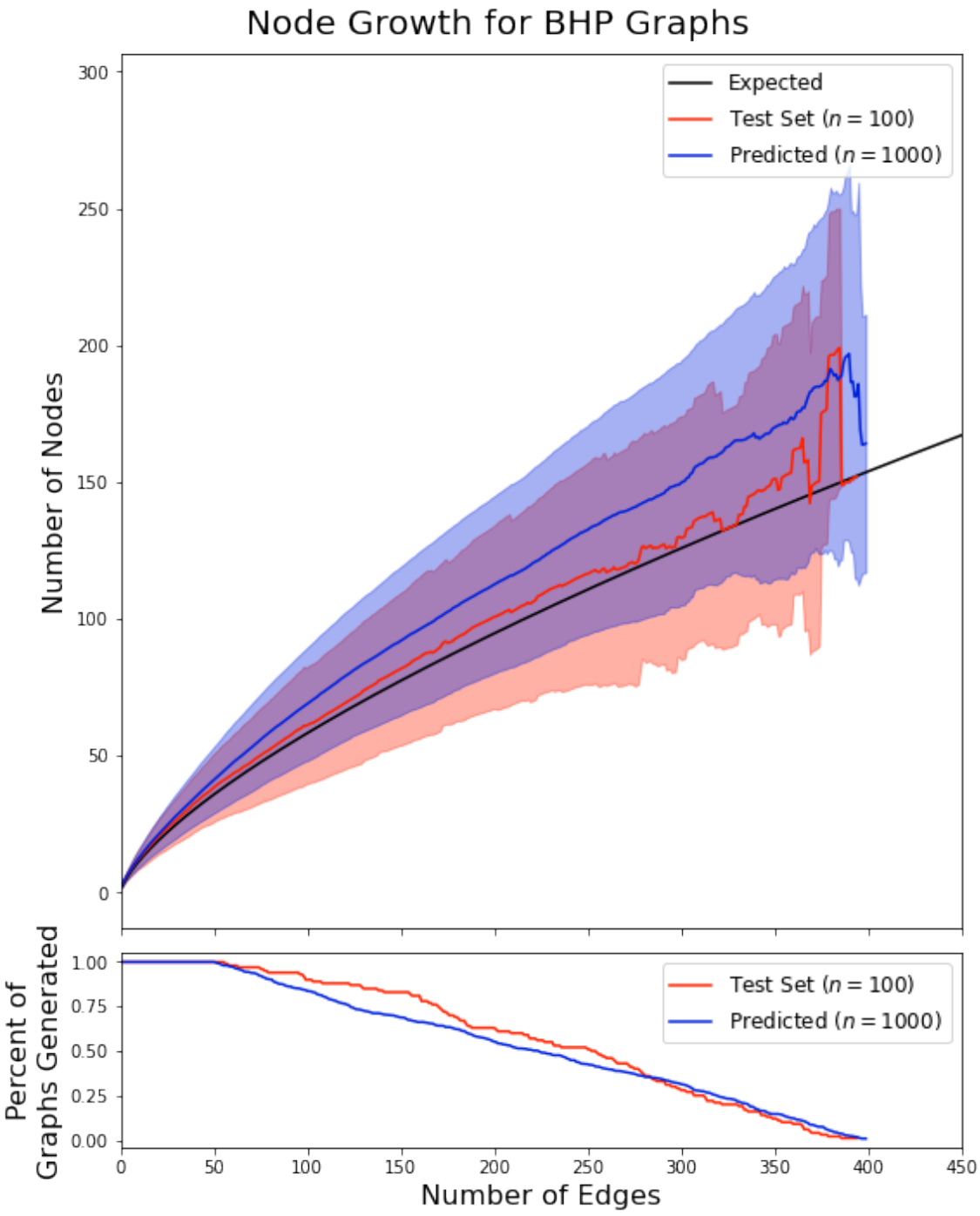

Figure 5.3: Expected and empirical node growth for binary Hollywood process graph sequences and generated graph sequences.

expected number of vertices in BHP graphs (see Crane and Dempsey (2016)):

$$E(|V_n|) \sim \frac{\Gamma(\sigma+1)}{\sigma\Gamma(\sigma+\alpha)}(2n)^{\sigma}. \tag{5.8}$$

Thus, our model is empirically able to generate sparse graphs, validating the intuition of §5.3.1.

5.4.2 Datasets

We evaluated our method on several synthetic graph generation models, specifically, cycle graphs, ladder graphs, Barabási-Albert graphs, and dynamic nonparametric network distribution (DNND, Ghalebi et al., 2019) graphs. In all cases, we generated 900 graphs for training, 100 for validating, and 100 for testing.

For the cycle, ladder, and Barabási-Albert graphs, we generated graphs with 10 to 40 nodes (selecting the number of nodes uniformly at random, with only even-sized ladder graphs). Barabási-Albert graphs were generated with an initial seed graph of 4 unconnected nodes, with each new node connecting to 4 unique nodes selected from the previously-generated nodes by preferential attachment (with probability of selection being proportional to node's degree). Cycle, ladder, and Barabási-Albert graphs were all generated using NetworkX (Hagberg et al., 2008).

DNND graphs are an extension to the edge-exchangeable models discussed that incorporates clustering and temporal dynamics, while maintaining sparsity. In our experiments, DNND graphs were all generated with 200 edges, allowing self-loops and duplicate edges, with parameters $\alpha = 1, \gamma = 1, \tau = 0.2, \sigma = 0.7$ and a node decay window of 20 edges and no cluster decay, see Ghalebi et al. (2019).

We also used several real-world temporal datasets from TUDataset (Morris et al., 2020), specifically a subset of the datasets used in Oettershagen et al. (2020). In particular, we used the DBLP, Facebook, Highschool, Infectious, MIT, and Tumblr datasets.

Dataset	Graphs	Average Nodes	Average Edges	Average Repeated Edges
Synthetic				
BA	1000	24.98	83.91	0
Cycles	1000	24.98	24.98	0
DNND	1000	18.44	200	180.00
Ladders	1000	24.97	35.45	0
Real				
DBLP	755	52.87	320.10	220.32
Facebook	995	95.72	269.02	167.30
Highschool	180	52.32	544.81	432.73
Infectious	200	50	459.72	333.74
MIT	97	20	1469.15	1432.55
Tumblr	373	53.11	199.79	128.16

Table 5.1: Properties of datasets used in our experiments.

A summary of synthetic and real dataset properties is in Table 5.1. For synthetic datasets, we report the properties for the combined training and validation subsets. For real-world datasets, we report the properties of combined training, validation and test subsets.

5.4.3 Implementation Details

In the quantitative experiments below, when generating new graphs, our method selects a training graph at random and generates a new graph with the same timestamps and number of edges. In the time-aware version of our model, these are the timestamps applied for constructing messages (see Equation 5.6). Furthermore, our method requires an edge ordering, so when a natural ordering is not available (such as for the Barabási-Albert graphs), we visit edges in a breadth-first search (BFS) starting at a randomly selected node. This choice of edge ordering based on BFS mirrors that of You et al. (2018) which constructs a node ordering based on BFS. The cycle and ladder graphs employed a natural ordering of edges (traversing the cycle and moving "up" the ladder, respectively), while DNND and the real-world datasets have an explicit time ordering.

To facilitate a direct comparison with our model, we modified the DeepGMG model to produce graphs with a pre-determined number of nodes, rather than using its prediction function of whether to continue adding nodes. For a fair comparison with our method, graphs generated by DeepGMG were constructed to select a training graph at random and generate a new graph with the same number of nodes.

For both methods, we use early stopping based on the validation set log-likelihood not improving for 30 epochs, considering graphs generated with that best set of model parameters.

5.4.4 Evaluation Metrics

To compare methods, we compute graph statistics for the generated graphs and compare against a held-out test set of graphs. Specifically, we seek to compare how well graphs generated from different methods match the test set in terms of the degree distribution, the local clustering coefficient distribution, the Laplacian spectrum distribution, and the graph density distribution (for background on these statistics, see §C.3.1). For the first three graph statistics, we will make comparisons using the maximum mean discrepancy (MMD, Gretton et al., 2012), a kernel-based distributional distance (see §C.1).

To calculate the MMD, we will use kernels based on the total variation distance:

$$k(x, y; \sigma) = \exp\left(-\frac{d_{\mathrm{TV}}(x, y)}{2\sigma^2}\right) = \exp\left(-\frac{1}{2\sigma^2}\frac{1}{2}\sum_{i=1}^{d}|x_i - y_i|\right),$$

with kernel parameters as suggested by O'Bray et al. (2021). In particular, for each graph, we will compute histograms for the degree distribution, local clustering coefficient distribution, and Laplacian spectrum distribution with the kernel's scale and histogram's number of bins shown in Table 5.2.

For the graph density, we compute the Kolmogorov-Smirnov statistic, which is the maximum absolute difference in the empirical CDFs of the density between the generated graphs and test graphs.

Graph Statistic	σ	Number of bins
Degree	1.0	Maximum degree of any graph
Local clustering coefficient	0.1	100
Laplacian spectrum	1.0	200

Table 5.2: Kernel and histogram parameters used in calculating MMD values for each type of distribution.

5.4.5 Comparison with DeepGMG

We now turn to a quantitative comparison with DeepGMG – a generative GNN that generates graphs node-by-node – to explore the advantage of an edge-based representation. We will compare both versions of our model with DeepGMG: the time-aware version that uses Equation 5.6 to construct messages ("time"), and the un-weighted version that uses Equation 5.5 to construct messages ("no time"). Using models trained as described in §5.4.3 we then generated novel graphs, and compared these novel graphs to those of the test sets using the metrics described above.

For DeepGMG models, we generated 100 novel graphs. For our models, we generated a number of novel graphs equal to the size of the training set. We repeated the training process for 4 different random seeds for each combination of model and training set unless otherwise noted. Finally, when computing the metrics discussed above, we treat ground truth and generated graphs as simple, removing self-loops and repeated edges.

Table 5.3 presents results on the synthetic datasets discussed above, comparing graphs generated from each method with those in the test set. We observe that overall, our method tends to produce graphs that are more similar to the test set than the graphs generated by DeepGMG. In particular, our method appears to better replicate the density. This matches the intuition developed in §5.3.1 and the empirical sparsity results in §5.4.1.

For the synthetic datasets, we also visualize randomly-selected graphs generated from DeepGMG and our model, in comparison to randomly-selected ground

Dataset + Metric	Ours (time)	Ours (no time)	DeepGMG
BA	$n = 4$	$n = 6$	$n = 4$
$\text{MMD}_{\text{degree}}$	0.35268 ± 0.00943	0.38978 ± 0.01611	$\mathbf{0.22016 \pm 0.01586}$
$\text{MMD}_{\text{cluster}}$	$\mathbf{0.10822 \pm 0.00065}$	0.11020 ± 0.00117	0.14204 ± 0.00011
$\text{MMD}_{\text{spectral}}$	$\mathbf{0.19653 \pm 0.00923}$	0.23202 ± 0.01701	0.21706 ± 0.00332
$\text{KS}_{\text{density}}$	$\mathbf{0.54475 \pm 0.02572}$	0.67117 ± 0.02720	0.89250 ± 0.02161
Cycles	$n = 4$	$n = 6$	$n = 5$
$\text{MMD}_{\text{degree}}$	0.08389 ± 0.00018	$\mathbf{0.08350 \pm 0.00025}$	0.12930 ± 0.01692
$\text{MMD}_{\text{cluster}}$	$\mathbf{0.00000 \pm 0.00000}$	0.00531 ± 0.00232	0.18182 ± 0.06401
$\text{MMD}_{\text{spectral}}$	0.14965 ± 0.00069	$\mathbf{0.14872 \pm 0.00146}$	0.29722 ± 0.00620
$\text{KS}_{\text{density}}$	$\mathbf{0.09850 \pm 0.00043}$	0.09983 ± 0.00146	0.81000 ± 0.04195
DNND	$n = 4$	$n = 5$	$n = 4$
$\text{MMD}_{\text{degree}}$	0.37253 ± 0.00360	0.33253 ± 0.01476	$\mathbf{0.09297 \pm 0.00482}$
$\text{MMD}_{\text{cluster}}$	0.39874 ± 0.00749	$\mathbf{0.39486 \pm 0.00724}$	0.50634 ± 0.01349
$\text{MMD}_{\text{spectral}}$	0.25171 ± 0.00781	0.22512 ± 0.01368	$\mathbf{0.17333 \pm 0.00211}$
$\text{KS}_{\text{density}}$	0.50975 ± 0.04510	$\mathbf{0.41720 \pm 0.03059}$	0.99250 ± 0.00217
Ladders	$n = 4$	$n = 6$	$n = 5$
$\text{MMD}_{\text{degree}}$	0.00397 ± 0.00043	$\mathbf{0.00375 \pm 0.00013}$	0.10733 ± 0.00009
$\text{MMD}_{\text{cluster}}$	$\mathbf{0.00000 \pm 0.00000}$	0.00000 ± 0.00000	0.00143 ± 0.00128
$\text{MMD}_{\text{spectral}}$	0.05223 ± 0.00027	$\mathbf{0.05133 \pm 0.00030}$	0.50509 ± 0.00086
$\text{KS}_{\text{density}}$	0.08125 ± 0.00134	$\mathbf{0.08033 \pm 0.00122}$	0.96000 ± 0.00000

Table 5.3: Synthetic dataset results, showing the mean and standard error of each metric on four trials unless otherwise noted.

truth test graphs in Figure 5.4. Overall, we see that we are able to capture the structure of each graph family (as does DeepGMG).

Table 5.4 presents results on the real-world datasets. We observe that, despite the success observed on synthetic graph datasets, our approach seems not to outperform DeepGMG on real-world graphs[3]. We hypothesize that this may be due to our method propagating messages after every edge (including repeat edges) and sending messages over repeated edges as well, resulting in many more messages sent in our method as compared to DeepGMG. In the graph convolutional network setting, it is known that too many convolutional layers can lead to over-smoothing, causing embeddings for different nodes to become uninformative or indistinguishable (Li et al., 2018a). The real-world datasets explored here are much larger than the synthetic graph datasets we explored; we suspect that over-smoothing may become a problem for our model as the graph size grows. We do note that the version of our method with an explicit dependence on timestamps does tend to outperform the version that does not weight messages with respect to time on the real-world datasets, though not always.

5.5 Discussion and Future Work

Here, we introduced our edge-based generative graph neural network. While the method performed well on synthetic data, it struggles with some aspects of real-world graphs. We note that DeepGMG does well in these settings. We suspect this is due to the increase in the number of messages sent due to repeated edges in our model as compared to DeepGMG, which may have caused a smoothing of embeddings that may have caused problems with the larger real-world datasets we considered. In preliminary work, we explored ways to avoid this in the future, such as reinitializing the node embeddings periodically, but found this not to improve performance. While

[3]Note that some entries of the table have fewer repetitions. In practice, we found that several combinations of model and dataset experienced numerical issues.

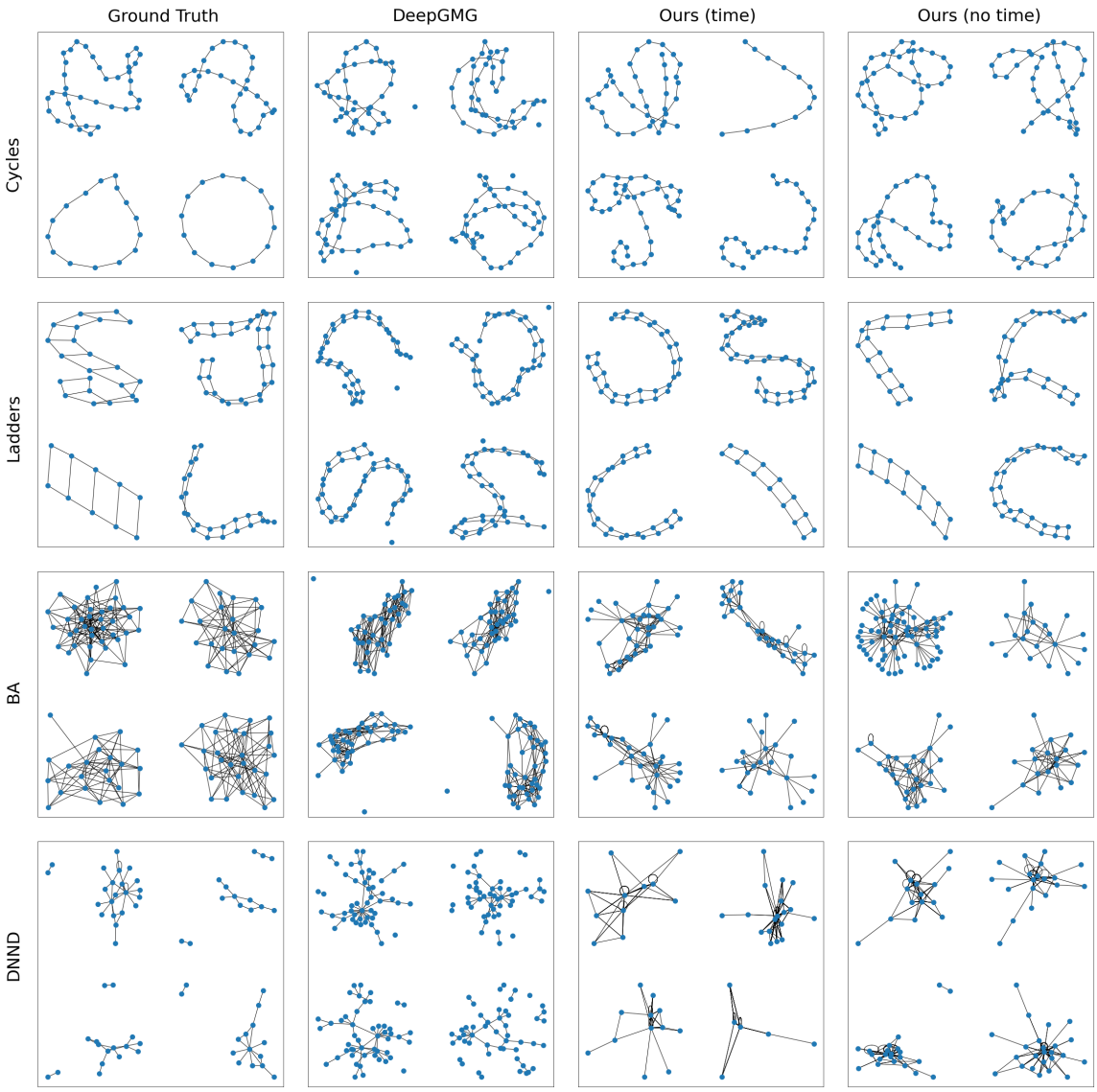

Figure 5.4: Comparison of generated graphs.

Dataset + Metric	Ours (time)	Ours (no time)	DeepGMG
DBLP	$n = 4$	$n = 6$	$n = 4$
$\text{MMD}_{\text{degree}}$	0.38128 ± 0.00871	0.41075 ± 0.00616	$\mathbf{0.06716 \pm 0.00766}$
$\text{MMD}_{\text{cluster}}$	0.23977 ± 0.00006	$\mathbf{0.23971 \pm 0.00001}$	0.26312 ± 0.01346
$\text{MMD}_{\text{spectral}}$	0.43224 ± 0.00550	0.43172 ± 0.00813	$\mathbf{0.15566 \pm 0.00464}$
$\text{KS}_{\text{density}}$	0.96535 ± 0.01185	0.96659 ± 0.00614	$\mathbf{0.47656 \pm 0.07952}$
Facebook	$n = 4$	$n = 6$	$n = 3$
$\text{MMD}_{\text{degree}}$	0.28136 ± 0.01842	0.36798 ± 0.01197	$\mathbf{0.02309 \pm 0.00195}$
$\text{MMD}_{\text{cluster}}$	0.80871 ± 0.06119	0.87979 ± 0.00490	$\mathbf{0.30354 \pm 0.00271}$
$\text{MMD}_{\text{spectral}}$	0.24639 ± 0.01206	0.39674 ± 0.02369	$\mathbf{0.07071 \pm 0.00148}$
$\text{KS}_{\text{density}}$	0.96442 ± 0.02603	0.99933 ± 0.00045	$\mathbf{0.36117 \pm 0.01849}$
Highschool	$n = 4$	$n = 4$	$n = 3$
$\text{MMD}_{\text{degree}}$	0.31491 ± 0.00859	0.39204 ± 0.01191	$\mathbf{0.19257 \pm 0.01318}$
$\text{MMD}_{\text{cluster}}$	$\mathbf{0.21266 \pm 0.00165}$	0.21684 ± 0.00106	0.21780 ± 0.00306
$\text{MMD}_{\text{spectral}}$	$\mathbf{0.25285 \pm 0.01048}$	0.34389 ± 0.01671	0.29707 ± 0.02010
$\text{KS}_{\text{density}}$	0.84394 ± 0.01469	0.76664 ± 0.06818	$\mathbf{0.23444 \pm 0.04132}$
Infectious	$n = 4$	$n = 4$	$n = 0$
$\text{MMD}_{\text{degree}}$	0.44871 ± 0.02280	$\mathbf{0.40831 \pm 0.00882}$	–
$\text{MMD}_{\text{cluster}}$	0.20164 ± 0.00004	$\mathbf{0.20136 \pm 0.00011}$	–
$\text{MMD}_{\text{spectral}}$	0.39669 ± 0.01661	$\mathbf{0.35226 \pm 0.01277}$	–
$\text{KS}_{\text{density}}$	0.90475 ± 0.03048	$\mathbf{0.80250 \pm 0.04155}$	–
MIT	$n = 0$	$n = 2$	$n = 2$
$\text{MMD}_{\text{degree}}$	–	0.64142 ± 0.01921	$\mathbf{0.09543 \pm 0.00096}$
$\text{MMD}_{\text{cluster}}$	–	0.80641 ± 0.00130	$\mathbf{0.63332 \pm 0.00997}$
$\text{MMD}_{\text{spectral}}$	–	0.47489 ± 0.04091	$\mathbf{0.09277 \pm 0.00473}$
$\text{KS}_{\text{density}}$	–	$\mathbf{0.82450 \pm 0.08733}$	0.86000 ± 0.02121
Tumblr	$n = 4$	$n = 5$	$n = 2$
$\text{MMD}_{\text{degree}}$	0.21777 ± 0.03324	0.33572 ± 0.01402	$\mathbf{0.06708 \pm 0.00416}$
$\text{MMD}_{\text{cluster}}$	0.54088 ± 0.01591	0.57516 ± 0.00149	$\mathbf{0.35256 \pm 0.01699}$
$\text{MMD}_{\text{spectral}}$	0.18212 ± 0.02233	0.27061 ± 0.01587	$\mathbf{0.14842 \pm 0.00133}$
$\text{KS}_{\text{density}}$	$\mathbf{0.74173 \pm 0.05625}$	0.86281 ± 0.02615	0.98324 ± 0.01185

Table 5.4: Real-world dataset results, showing the mean and standard error of each metric on four trials unless otherwise noted.

the time-weighted version of our model could theoretically discount messages from nodes that were not recently communicated with, we could make this more explicit by only sending messages over edges within a certain window of time from the current timestamp.

Appendix A: Supplemental Material for Spike-and-Slab Probabilistic Backpropagation[1]

A.1 Approximating Messages Using a Spike-and-Slab Distribution

We take the following for our approximating distribution q, that is, the spike and (non-central) slab:

$$\widetilde{q}(z; \widetilde{\rho}, \widetilde{m}, \widetilde{v}) = (1 - \widetilde{\rho})\delta_0(z) + \widetilde{\rho}\mathrm{N}(z; \widetilde{m}, \widetilde{v}).$$

We seek to minimize $KL(p\|\widetilde{q})$ with respect to $\widetilde{\rho}, \widetilde{m}, \widetilde{v}$. Were $\widetilde{q}$ a Gaussian, this would be solved by moment matching $\widetilde{m} = \mathbb{E}_p[X]$ and $\widetilde{v} = \mathbb{V}_p[X]$ (Minka, 2001). Here, we derive the appropriate values of $\widetilde{\rho}, \widetilde{m}, \widetilde{v}$. For simplicity, and since we make use of a mean-field approximation, we'll focus on a univariate version, similar to PBP's approach:

$$\min_{\widetilde{q}} KL(p\|\widetilde{q}) \propto -\int_{\mathbb{R}} p(z) \log(\widetilde{q}(z))dz = -\int_{\mathbb{R}} p(z) \log\left((1 - \widetilde{\rho})\delta_0(z) + \widetilde{\rho}\mathrm{N}(z; \widetilde{m}, \widetilde{v})\right) dz.$$

We begin by giving values for $\widetilde{m}$, $\widetilde{v}$ and $\widetilde{\rho}$ in terms of the mean, variance, and probability at zero of the distribution being approximated.

A.1.1 Approximating a Distribution with Known Mean, Variance, and Probability of Zero

A.1.1.1 Slab Mean Parameter $\widetilde{m}$

We begin by seeking to minimize $KL(p\|\widetilde{q})$ with respect to the slab's mean parameter, $\widetilde{m}$:

[1]This appendix will appear appeared as the supplemental material for Evan Ott and Sinead Williamson. Spike-and-Slab Probabilistic Backpropagation: When Smarter Approximations Make No Difference. In *I Can't Believe It's Not Better Workshop: Understanding Deep Learning Through Empirical Falsification*, 2022b. URL `https://openreview.net/forum?id=iYAdBHSA_Pt`.

$$
\begin{aligned}
-\frac{d}{d\widetilde{m}}\int_{\mathbb{R}} p(z)\log(\widetilde{q}(z))dz &= -\int_{\mathbb{R}} p(z)\frac{\frac{\partial}{\partial m}\widetilde{q}(z;\widetilde{\rho},\widetilde{m},\widetilde{v})}{\widetilde{q}(z;\rho,\widetilde{m},\widetilde{v})}dz \\
&= -\int_{\mathbb{R}} p(z)\frac{\frac{\partial}{\partial\widetilde{m}}\left((1-\widetilde{\rho})\delta_0(z)+\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})\right)}{(1-\widetilde{\rho})\delta_0(z)+\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})}dz \\
&= -\int_{\mathbb{R}} p(z)\frac{\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})\left(\frac{z-\widetilde{m}}{\widetilde{v}}\right)+0}{(1-\widetilde{\rho})\delta_0(z)+\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})}dz \\
&= -\int_{\mathbb{R}} p(z)\frac{\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})\left(\frac{z-\widetilde{m}}{\widetilde{v}}\right)+(1-\widetilde{\rho})\delta_0(z)\left(\frac{z-\widetilde{m}}{\widetilde{v}}\right)-(1-\widetilde{\rho})\delta_0(z)\left(\frac{z-\widetilde{m}}{\widetilde{v}}\right)}{(1-\widetilde{\rho})\delta_0(z)+\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})}dz \\
&= -\int_{\mathbb{R}} p(z)\left(\frac{z-\widetilde{m}}{\widetilde{v}}\right)dz+\int_{\mathbb{R}} p(z)\frac{(1-\widetilde{\rho})\delta_0(z)\left(\frac{z-\widetilde{m}}{\widetilde{v}}\right)}{(1-\widetilde{\rho})\delta_0(z)+\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})}dz \\
&= -\mathbb{E}_p\left[\frac{Z-\widetilde{m}}{\widetilde{v}}\right]+\int_{\mathbb{R}} p(z)\frac{(1-\widetilde{\rho})\delta_0(z)\left(\frac{z-\widetilde{m}}{\widetilde{v}}\right)}{(1-\widetilde{\rho})\delta_0(z)+\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})}dz.
\end{aligned}
\tag{A.1}
$$

The integral in the final line of Equation A.1 is nontrivial. We approximate this term by considering the delta function as a limit of the uniform distribution $u_a(z) = \mathrm{Unif}(z;[-1/2a,1/2a])$, such that as $a\to\infty$, we recover the delta function at 0 $\delta_0(z)$. Because u_a is 0 outside the range $[-1/2a,1/2a]$, we can restrict the domain of the integral:

$$
\begin{aligned}
I &= \lim_{a\to\infty}\int_{\mathbb{R}} p(z)\frac{(1-\widetilde{\rho})u_a(z)}{(1-\widetilde{\rho})u_a(z)+\rho\mathrm{N}(z;\widetilde{m},\widetilde{v})}f(z)dz \\
&= \lim_{a\to\infty}\int_{-1/2a}^{1/2a} p(z)\frac{(1-\widetilde{\rho})a}{(1-\widetilde{\rho})a+\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v})}f(z)dz.
\end{aligned}
$$

Under the assumption that $(1-\widetilde{\rho})a \gg \rho\mathrm{N}(z;\widetilde{m},\widetilde{v})$ (as $a\to\infty$), we have

$$
I \approx \lim_{a\to\infty}\int_{-1/2a}^{1/2a} p(z)\frac{(1-\widetilde{\rho})a}{(1-\widetilde{\rho})a}f(z)dz = \lim_{\epsilon\to 0^+}\int_{-\epsilon}^{\epsilon} p(z)f(z)dz.
$$

Setting the derivative in Equation A.1 to zero, we have

$$
\begin{aligned}
\mathbb{E}_p[Z] &\approx \widetilde{m}+\lim_{\epsilon\to 0^+}\int_{-\epsilon}^{\epsilon} p(z)\,(z-\widetilde{m})\,dz \\
&= \widetilde{m}-\widetilde{m}\lim_{\epsilon\to 0^+}\int_{-\epsilon}^{\epsilon} p(z)dz+\lim_{\epsilon\to 0^+}\int_{-\epsilon}^{\epsilon} p(z)zdz \\
&= \widetilde{m}-\widetilde{m}\mathbb{P}_p\left[Z=0\right]+0.
\end{aligned}
$$

This yields our solution

$$\widetilde{m} = \frac{\mathbb{E}_p[Z]}{1 - \mathbb{P}_p[Z = 0]}.$$

A.1.1.2 Slab Variance Parameter $\widetilde{v}$

For the variance parameter $\widetilde{v}$, we similarly obtain:

$$\begin{aligned}
\frac{\partial \widetilde{q}(z; \widetilde{\rho}, \widetilde{m}, \widetilde{v})}{\partial \widetilde{v}} &= \widetilde{\rho} \frac{1}{\sqrt{2\pi\widetilde{v}}} \exp\left(-\frac{(z-\widetilde{m})^2}{2\widetilde{v}}\right)\left(\frac{(z-\widetilde{m})^2 - \widetilde{v}}{2\widetilde{v}^2}\right) \\
&= \widetilde{\rho}\mathrm{N}(z; \widetilde{m}, \widetilde{v})\left(\frac{(z-\widetilde{m})^2 - \widetilde{v}}{2\widetilde{v}^2}\right) \\
0 = -\frac{d}{d\widetilde{v}} \int_{\mathbb{R}} p(z) \log(\widetilde{q}(z)) dz &= -\int_{\mathbb{R}} p(z) \frac{\frac{\partial}{\partial \widetilde{v}} \widetilde{q}(z; \widetilde{\rho}, \widetilde{m}, \widetilde{v})}{\widetilde{q}(z; \widetilde{\rho}, \widetilde{m}, \widetilde{v})} dz \\
&= -\int_{\mathbb{R}} p(z) \frac{\widetilde{\rho}\mathrm{N}(z; \widetilde{m}, \widetilde{v})\left(\frac{(z-\widetilde{m})^2 - \widetilde{v}}{2\widetilde{v}^2}\right)}{(1-\widetilde{\rho})\delta_0(z) + \widetilde{\rho}\mathrm{N}(z; \widetilde{m}, \widetilde{v})} dz \\
&= -\mathbb{E}_p\left[\frac{(Z-\widetilde{m})^2 - \widetilde{v}}{2\widetilde{v}^2}\right] + \int_{\mathbb{R}} p(z) \frac{(1-\widetilde{\rho})\delta_0(z)\left(\frac{(z-\widetilde{m})^2 - \widetilde{v}}{2\widetilde{v}^2}\right)}{(1-\widetilde{\rho})\delta_0(z) + \widetilde{\rho}\mathrm{N}(z; \widetilde{m}, \widetilde{v})} dz \\
&\approx -\mathbb{E}_p\left[\frac{(Z-\widetilde{m})^2 - \widetilde{v}}{2\widetilde{v}^2}\right] + \lim_{\epsilon \to 0^+} \int_{-\epsilon}^{\epsilon} p(z)\left(\frac{(z-\widetilde{m})^2 - \widetilde{v}}{2\widetilde{v}^2}\right) dz.
\end{aligned}$$

Thus, we have

$$\begin{aligned}
\mathbb{E}_p[Z^2] - 2\widetilde{m}\mathbb{E}_p[Z] + \widetilde{m}^2 \approx \widetilde{v}&\left(1 - \lim_{\epsilon \to 0^+} \int_{-\epsilon}^{\epsilon} p(z) dz\right) \\
&+ \lim_{\epsilon \to 0^+} \int_{-\epsilon}^{\epsilon} p(z)(z^2 - 2z\widetilde{m} + \widetilde{m}^2) dz \\
&= \widetilde{v}(1 - \mathbb{P}_p[Z = 0]) + \widetilde{m}^2 \mathbb{P}_p[Z = 0] + 0.
\end{aligned}$$

This yields

$$\begin{aligned}
\widetilde{v} &= \frac{\mathbb{E}_p[Z^2] - 2\widetilde{m}\mathbb{E}_p[Z] + \widetilde{m}^2(1 - \mathbb{P}_p[Z = 0])}{1 - \mathbb{P}_p[Z = 0]} \\
&= \frac{1}{1 - \mathbb{P}_p[Z = 0]}\left(\mathbb{V}_p[Z] - \frac{\mathbb{E}_p[Z]^2 \mathbb{P}_p[Z = 0]}{1 - \mathbb{P}_p[Z = 0]}\right).
\end{aligned}$$

A.1.1.3 Slab Probability Parameter $\widetilde{\rho}$

Finally, for the mass of the slab $\widetilde{\rho}$, we have

$$\begin{aligned}
\frac{\partial \widetilde{q}(z;\widetilde{\rho},\widetilde{m},\widetilde{v})}{\partial \widetilde{\rho}} &= \mathrm{N}(z;\widetilde{m},\widetilde{v}) - \delta_0(z) \\
0 = -\frac{d}{d\widetilde{\rho}} \int_{\mathbb{R}} p(z) \log(\widetilde{q}(z)) dz &= -\int_{\mathbb{R}} p(z) \frac{\frac{\partial}{\partial \widetilde{\rho}} q(z;\widetilde{\rho},\widetilde{m},\widetilde{v})}{\widetilde{q}(z;\widetilde{\rho},\widetilde{m},\widetilde{v})} dz \\
&= -\int_{\mathbb{R}} p(z) \frac{\mathrm{N}(z;\widetilde{m},\widetilde{v}) - \delta_0(z)}{\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v}) + (1-\widetilde{\rho})\delta_0(z)} dz \\
&= -\int_{\mathbb{R}} p(z) \frac{1}{\widetilde{\rho}} \frac{\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v}) - \widetilde{\rho}\delta_0(z) - \delta_0(z) + \delta_0(z)}{\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v}) + (1-\widetilde{\rho})\delta_0(z)} dz \\
&= -\frac{1}{\widetilde{\rho}} \int_{\mathbb{R}} p(z) dz + \frac{1}{\widetilde{\rho}} \int_{\mathbb{R}} p(z) \frac{\delta_0(z)}{\widetilde{\rho}\mathrm{N}(z;\widetilde{m},\widetilde{v}) + (1-\widetilde{\rho})\delta_0(z)} dz \\
&\approx -\frac{1}{\widetilde{\rho}} + \frac{1}{\widetilde{\rho}} \lim_{\epsilon \to 0^+} \int_{-\epsilon}^{\epsilon} p(z) \frac{1}{1-\widetilde{\rho}} dz \\
&= -\frac{1}{\widetilde{\rho}} + \frac{1}{\widetilde{\rho}(1-\widetilde{\rho})} \mathbb{P}_p[Z=0],
\end{aligned}$$

yielding the simple and intuitive result:

$$\widetilde{\rho} = 1 - \mathbb{P}_p[Z=0].$$

A.1.1.4 Connection to Moment Matching

As it turns out, minimizing $KL(p\|q)$ for the spike-and-slab turns out to be equivalent to matching the slab probability and then matching the first and second moments:

$$\begin{aligned}
\mathbb{P}_{\widetilde{q}}[Z=0] &= 1-\widetilde{\rho} \\
\mathbb{E}_{\widetilde{q}}[Z] &= \widetilde{\rho}\widetilde{m} \\
\mathbb{V}_{\widetilde{q}}[Z] &= \mathbb{E}_{\widetilde{q}}\left[Z^2\right] - \mathbb{E}_{\widetilde{q}}[Z]^2 \\
&= \widetilde{\rho}(\widetilde{m}^2 + \widetilde{v}) - \widetilde{\rho}^2\widetilde{m}^2.
\end{aligned}$$

Substituting in the values of $(\widetilde{\rho}, \widetilde{m}, \widetilde{v})$ obtained above in the KL-minimization, we find that we yield the highly intuitive result:

$$\mathbb{P}_p[Z=0] = \mathbb{P}_{\widetilde{q}}[Z=0], \qquad \mathbb{E}_p[Z] = \mathbb{E}_{\widetilde{q}}[Z], \qquad \mathbb{V}_p[Z] = \mathbb{V}_{\widetilde{q}}[Z].$$

A.1.2 Message Parameters Following a Linear Layer

Above, we established the values of parameters $(\widetilde{\rho}, \widetilde{m}, \widetilde{v})$ in terms of properties of the distribution we intend to approximate. We now shift our focus to deriving the specific values for the transformations that occur within spike-and-slab PBP, namely a linear combination and a ReLU. We begin with the linear combination $z_j^{(\ell,\text{linear})} = \sum_i z_i^{(\ell-1)} w_{ij}$, seeking the values of $\mathbb{E}_p[z_j^{(\ell,\text{linear}}]$, $\mathbb{V}_p[z_j^{(\ell,\text{linear}}]$, and $\mathbb{P}_p[z_j^{(\ell,\text{linear}} = 0]$ under the assumption that

$$\begin{aligned} z_i^{(\ell-1)} &\overset{\text{ind}}{\sim} (1-\widetilde{\rho}_i^{(\ell-1)})\delta_0 + \widetilde{\rho}^{(\ell-1)} \mathrm{N}(\widetilde{m}^{(\ell-1)}, \widetilde{v}^{(\ell-1)}) \\ w_{ij} &\overset{\text{ind}}{\sim} \mathrm{N}(m_{ij}, v_{ij}). \end{aligned}$$

From this, we can calculate the mean and variance

$$\begin{aligned} \mathbb{E}\left[z_j^{(\ell,\text{linear})}\right] &= \sum_i (\widetilde{\rho}_i^{(\ell-1)} \widetilde{m}_i^{(\ell)}) m_{ij} \\ \mathbb{V}\left[z^{(\ell,\text{linear})}\right] &= \sum_i \widetilde{\rho}_i^{(\ell-1)} (\widetilde{m}_i^{(\ell-1)})^2 v_{ij} + \widetilde{\rho}_i^{(\ell-1)} \widetilde{v}_i^{(\ell-1)} m_{ij}^2 \\ &\quad + \widetilde{\rho}_i^{(\ell-1)} \widetilde{v}_i^{(\ell-1)} v_{ij} + \widetilde{\rho}_i^{(\ell-1)} (1-\widetilde{\rho}_i^{(\ell-1)}) (\widetilde{m}_i^{(\ell-1)})^2 m_{ij}^2. \end{aligned}$$

Now, incorporating the rescaling transformation used by Hernández-Lobato and Adams (2015), $\mathbf{z}^{(\ell,\text{linear})} = \mathbf{W}^{(\ell)} \mathbf{z}^{(\ell-1)} / \sqrt{n_{\ell-1}+1}$, we have

$$\begin{aligned} \mathbb{E}\left[\mathbf{z}^{(\ell,\text{linear})}\right] &= \mathbf{M}^{(\ell)} \left(\widetilde{\rho}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)}\right) / \sqrt{n_{\ell-1}+1} \\ \mathbb{V}\left[\mathbf{z}^{(\ell,\text{linear})}\right] &= \Big[\mathbf{V}^{(\ell)} \left(\widetilde{\rho}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)}\right) + \left(\mathbf{M}^{(\ell)} \circ \mathbf{M}^{(\ell)}\right) \left(\widetilde{\rho}^{(\ell-1)} \circ \widetilde{\mathbf{v}}^{(\ell-1)}\right) \\ &\quad + \left(\mathbf{M}^{(\ell)} \circ \mathbf{M}^{(\ell)}\right) \left(\widetilde{\rho}^{(\ell-1)} \circ \left(1-\widetilde{\rho}^{(\ell-1)}\right) \circ \widetilde{\mathbf{m}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)}\right) \\ &\quad + \mathbf{V}^{(\ell)} \left(\widetilde{\rho}^{(\ell-1)} \circ \widetilde{\mathbf{v}}^{(\ell-1)}\right)\Big] / \left(n_{\ell-1}+1\right) \\ \mathbb{P}\left[\mathbf{z}^{(\ell,\text{linear})} = 0\right] &= \prod_i (1-\widetilde{\rho}_i^{(\ell-1)}). \end{aligned}$$

Combining these moments with the general forms derived earlier, we have the following equations (replacing Equations 13-14 of the original PBP method) for the distribution of linear layers in SSPBP with parameters $\widetilde{\boldsymbol{\rho}}^{(\ell,\text{linear})}, \widetilde{\mathbf{m}}^{(\ell,\text{linear})}, \widetilde{\mathbf{v}}^{(\ell,\text{linear})}$:

$$\begin{aligned}
\widetilde{\rho}_j^{(\ell,\text{linear})} &= 1 - \prod_i (1 - \widetilde{\rho}_i^{(\ell-1)}) \\
\widetilde{\boldsymbol{\rho}}^{(\ell,\text{linear})} \circ \widetilde{\mathbf{m}}^{(\ell,\text{linear})} &= \frac{\mathbf{M}^{(\ell}(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)})}{\sqrt{n_{\ell-1}+1}} \\
\widetilde{\boldsymbol{\rho}}^{(\ell,\text{linear})} \circ \widetilde{\boldsymbol{\rho}}^{(\ell,\text{linear})} \circ \widetilde{\mathbf{v}}^{(\ell,\text{linear})} &= \widetilde{\boldsymbol{\rho}}^{(\ell,\text{linear})} \circ \mathbb{V}\left[z^{(\ell,\text{linear})}\right] - (1 - \widetilde{\boldsymbol{\rho}}^{(\ell,\text{linear})}) \mathbb{E}\left[\mathbf{z}^{(\ell,\text{linear})}\right]^2 \\
\widetilde{\boldsymbol{\rho}}^{(\ell,\text{linear})} \circ (n_{\ell-1}+1) \widetilde{\mathbf{v}}^{(\ell,\text{linear})} &= \\
&\quad \mathbf{V}^{(\ell)} \left(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)} \right) \\
&\quad + \left(\mathbf{V}^{(\ell)} + \mathbf{M}^{(\ell)} \circ \mathbf{M}^{(\ell)} \right) \left(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \widetilde{\mathbf{v}}^{(\ell-1)} \right) \\
&\quad + \left(\mathbf{M}^{(\ell)} \circ \mathbf{M}^{(\ell)} \right) \left(\widetilde{\boldsymbol{\rho}}^{(\ell-1)} \circ \left(1 - \widetilde{\boldsymbol{\rho}}^{(\ell-1)} \right) \circ \widetilde{\mathbf{m}}^{(\ell-1)} \circ \widetilde{\mathbf{m}}^{(\ell-1)} \right).
\end{aligned} \tag{A.2}$$

A.1.3 Message Parameters Following a Linear Layer and ReLU Activation

We now consider passing the messages from Equation A.2 through a ReLU, to get the parameters $(\widetilde{\boldsymbol{\rho}}^{(\ell,\text{ReLU})}, \widetilde{\mathbf{m}}^{(\ell,\text{ReLU})}, \widetilde{\mathbf{v}}^{(\ell,\text{ReLU})})$ of the resulting message.

To compute these parameters, it's useful to employ a hierarchical model:

$$\begin{aligned}
A_i &\sim (1 - \widetilde{\rho}_i^{(\ell,\text{linear})}) \delta_0 + \widetilde{\rho}_i^{(\ell,\text{linear})} \text{N}(\widetilde{m}^{(\ell,\text{linear})}, \widetilde{v}^{(\ell,\text{linear})}) \\
T_i &\sim \text{Ber}(\widetilde{\rho}_i^{(\ell,\text{linear})}) \\
A_i | T_i = 0 &\sim \delta_0 \\
A_i | T_i = 1 &\sim \text{N}(\widetilde{m}_i^{(\ell,\text{linear})}, \widetilde{v}_i^{(\ell,\text{linear})}) \\
B_i | A_i = a \ &= \ \text{ReLU}(a) = \max(a, 0),
\end{aligned}$$

implying

$$B_i \sim \left(1 - \widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\frac{\widetilde{m}_i^{(\ell,\text{linear})}}{\sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}}\right)\right)\delta_0 + \widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\frac{\widetilde{m}_i^{(\ell,\text{linear})}}{\sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}}\right) \text{TN}_{(0,\infty)}\left(\widetilde{m}_i^{(\ell,\text{linear})}, \widetilde{v}_i^{(\ell,\text{linear})}\right)$$

where TN indicates a truncated normal. Similarly, let's now introduce a hierarchy for B_i, along with a "dummy" variable X_i to make the notation below a little more straightforward:

$$\begin{aligned} L_i &\sim \text{Ber}\left(\widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\frac{\widetilde{m}_i^{(\ell,\text{linear})}}{\sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}}\right)\right) \\ B_i|L_i = 0 &\sim \delta_0 \\ X_i &\sim \text{TN}_{(0,\infty)}\left(\widetilde{m}_i^{(\ell,\text{linear})}, \widetilde{v}_i^{(\ell,\text{linear})}\right) \\ B_i|L_i = 1 &\sim \delta_{X_i}. \end{aligned}$$

Thus, $\mathbb{E}[X_i], \mathbb{V}[X_i]$, *etc.* correspond to the conditional expectation and variance $\mathbb{E}[B_i|L_i = 1], \mathbb{V}[B_i|L_i = 1]$.

We next compute the moments of $(\widetilde{\boldsymbol{\rho}}^{(\ell,\text{ReLU})}, \widetilde{\mathbf{m}}^{(\ell,\text{ReLU})}, \widetilde{\mathbf{v}}^{(\ell,\text{ReLU})})$ by the moment-matching we derived above:

$$\begin{aligned} \mathbb{P}[B_i = 0] =& \mathbb{P}[L_i = 0] = 1 - \widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\widetilde{m}_i^{(\ell,\text{linear})} \Big/ \sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}\right) \\ \mathbb{E}[B_i] =& \mathbb{E}\left[\mathbb{E}[B_i|L_i]\right] = \mathbb{P}[L_i = 0] \cdot 0 + \mathbb{P}[L_i = 1]\mathbb{E}[X_i] \\ =& \mathbb{P}[L_i = 1]\mathbb{E}[X_i] = \widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\widetilde{m}_i^{(\ell,\text{linear})} \Big/ \sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}\right) \mathbb{E}[X_i] \\ \mathbb{V}[B_i] =& \mathbb{E}[B_i^2] - \mathbb{E}[B_i]^2 = \mathbb{P}[L_i = 0] \cdot 0 + \mathbb{P}[L_i = 1]\mathbb{E}[X_i^2] - \left(\mathbb{P}[L_i = 1]\mathbb{E}[X_i]\right)^2 \\ =& \mathbb{P}[L_i = 1]\mathbb{E}[X_i^2] - \mathbb{P}[L_i = 1]^2\mathbb{E}[X_i]^2 \\ =& \widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\widetilde{m}_i^{(\ell,\text{linear})} \Big/ \sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}\right) \mathbb{E}[X^2] \\ & - \left(\widetilde{\rho}_i^{(\ell,\text{linear})}\right)^2 \Phi\left(\widetilde{m}_i^{(\ell,\text{linear})} \Big/ \sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}\right)^2 \mathbb{E}[X_i]^2. \end{aligned}$$

Using the spike-and-slab moment-matching results above, we obtain:

$$\begin{aligned}
\widetilde{\rho}_i^{(\ell,\text{ReLU})} &= 1 - \mathbb{P}[B_i = 0] \\
&\quad 1 - \left(1 - \widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\widetilde{m}_i^{(\ell,\text{linear})} \Big/ \sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}\right)\right) = \rho\Phi\left(\frac{m}{\sqrt{v}}\right) \\
\widetilde{m}_i^{(\ell,\text{ReLU})} &= \frac{\mathbb{E}[B_i]}{\widetilde{\rho}_i^{(\ell,\text{ReLU})}} = \frac{\widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\widetilde{m}_i^{(\ell,\text{linear})} \Big/ \sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}\right) \mathbb{E}[X_i]}{\widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\widetilde{m}_i^{(\ell,\text{linear})} \Big/ \sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}\right)} = \mathbb{E}[X_i] \\
\widetilde{v}_i^{(\ell,\text{ReLU})} &= \frac{\mathbb{V}[B_i] - \widetilde{\rho}_i^{(\ell,\text{ReLU})}\left(1 - \widetilde{\rho}_i^{(\ell,\text{ReLU})}\right)\left(\widetilde{m}_i^{(\ell,\text{ReLU})}\right)^2}{\widetilde{\rho}_i^{(\ell,\text{ReLU})}} = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 = \mathbb{V}[X_i].
\end{aligned}$$

In other words, we match the probability mass of the spike and match the mean and variance in our normal approximation to the mean and variance of the truncated normal! This yields our final message parameters,

$$\begin{aligned}
\widetilde{m}_i^{(\ell,\text{ReLU})} &= \widetilde{m}_i^{(\ell,\text{linear})} + \gamma_i \sqrt{\widetilde{v}_i^{(\ell,\text{linear})}} \\
\widetilde{v}_i^{(\ell,\text{ReLU})} &= \widetilde{v}_i^{(\ell,\text{linear})} \left(1 - \gamma_i \left(\gamma_i + \frac{\widetilde{m}_i^{(\ell,\text{linear})}}{\sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}}\right)\right) \\
\widetilde{\rho}_i^{(\ell,\text{ReLU})} &= \widetilde{\rho}_i^{(\ell,\text{linear})} \Phi\left(\frac{\widetilde{m}_i^{(\ell,\text{linear})}}{\sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}}\right) \\
\gamma_i &= \frac{\phi\left(\frac{\widetilde{m}_i^{(\ell,\text{linear})}}{\sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}}\right)}{\Phi\left(\frac{\widetilde{m}_i^{(\ell,\text{linear})}}{\sqrt{\widetilde{v}_i^{(\ell,\text{linear})}}}\right)}.
\end{aligned}$$

The intermediate parameter γ_i can be replaced with a "robust" version as appropriate, following (Hernández-Lobato and Adams, 2015).

A.1.4 Normalization Constant

The normalization constant Z of Hernández-Lobato and Adams (2015) in Equation 12 (and its uses elsewhere in the automatic differentiation for the model parameters) is modified slightly in our approach. As final output parameters, we produce $(\widetilde{\rho}^{(L)}, \widetilde{m}^{(L)}, \widetilde{v}^{(L)})$, along with the homoscedastic noise estimate $\beta^\gamma/(\alpha^\gamma - 1)$ (unchanged from PBP). As such, the relevant replacement for Equation 12 of (Hernández-Lobato and Adams, 2015) is

$$Z \approx (1 - \widetilde{\rho}^{(L)})\mathrm{N}(y_n|0, \beta^\gamma/(\alpha^\gamma - 1)) + \rho^{(L)}\mathrm{N}(y_n|\widetilde{m}^{(L)}, \beta^\gamma/(\alpha^\gamma - 1) + \widetilde{v}^{(L)}).$$

A.2 Proof of Equivalence Between PBP and SSPBP with a Bias Term

Here, we show that after the linear combination step, the models produce the same resultant distribution. Consider the following model for the output Y of a single node (for simplicity in notation) with random K-dimensional input $\mathbf{x}$, noting that this corresponds exactly to the first hidden layer's activation function and the second layer's linear combination step[2]:

$$\begin{aligned}
x_i &\sim \mathrm{N}(\widetilde{m}_i, \widetilde{v}_i), \quad i = 1:K \\
t_i|x_i &= \mathrm{ReLU}(x_i), \quad i = 1:K \\
t_{K+1} &\sim \delta_1 = \mathrm{N}(1, 0) \quad \leftarrow \text{the bias term} \\
w_i &\sim \mathrm{N}(m_i, v_i), \quad i = 1:K+1 \\
y|\mathbf{t}, \mathbf{w} &= \frac{1}{\sqrt{K+1}}\mathbf{w}^\top\mathbf{t}.
\end{aligned}$$

[2]The factor $1/\sqrt{K+1}$ "keeps the scale of the input to each neuron independent of the number of incoming connections." (Hernández-Lobato and Adams, 2015)

Following Hernández-Lobato and Adams (2015), we also assume

$$\begin{aligned}\alpha_i &= \frac{\widetilde{m}_i}{\sqrt{\widetilde{v}_i}} \\ \gamma_i &= \frac{\phi(\alpha_i)}{\Phi(\alpha_i)} \\ v_i' &= \widetilde{m}_i + \widetilde{v}_i\gamma_i.\end{aligned}$$

A.2.1 Parameters of the PBP Message

Under standard PBP, we model the distribution of t_i with a Gaussian with mean $m_i^{(t)}$ and variance $v_i^{(t)}$, where

$$\begin{aligned}m_i^{(t)} &= \Phi(\alpha_i)v_i', \quad i = 1:K, \qquad m_{K+1}^{(t)} = 1, \\ v_i^{(t)} &= (1-\Phi(\alpha_i))m_i^{(t)}v_i' + \Phi(\alpha_i)v_i(1-\gamma_i(\gamma_i+\alpha_i)), \quad i = 1:K, \\ &= \Phi(\alpha_i)(1-\Phi(\alpha_i))(v_i')^2 + \Phi(\alpha_i)v_i(1-\gamma_i(\gamma_i+\alpha_i)), \qquad v_{K+1}^{(t)} = 0.\end{aligned}$$

We then approximate the distribution of y using a Gaussian with mean $m^{(y)}$ and variance $v^{(y)}$, where

$$\begin{aligned}m^{(y)} &= \frac{1}{\sqrt{K+1}}\mathbf{m}^\top\mathbf{m}^{(t)} = \frac{1}{\sqrt{K+1}}\left(m_{K+1} + \sum_{i=1}^K m_i\Phi(\alpha_i)v_i'\right) \\ (K+1)v^{(y)} &= (\mathbf{m}\circ\mathbf{m}+\mathbf{v})^\top\mathbf{v}^{(t)} + \mathbf{v}^\top(\mathbf{m}^{(t)}\circ\mathbf{m}^{(t)}) \\ &= v_{K+1} + \sum_{i=1}^K (m_i^2+v_i)(1-\Phi(\alpha_i))\Phi(\alpha_i)(v_i')^2 \\ &\qquad + (m_i^2+v_i)\Phi(\alpha_i)\widetilde{v}_i(1-\gamma_i(\gamma_i+\alpha_i)) + v_i\Phi(\alpha_i)^2(v_i')^2.\end{aligned} \tag{A.3}$$

A.2.2 Parameters of the SSPBP Message

We repeat our analysis using spike-and-slab approximations, so the distribution of t_i is parameterized by $(\rho_i^{(t)}, m_i^{(t)}, v_i^{(t)})$, and the distribution of y is parameterized

by $(\rho^{(y)}, m^{(y)}, v^{(y)})$, such that

$$\begin{aligned}
\rho_i^{(t)} &= \Phi(\alpha_i), \quad i = 1:K, & \rho_{K+1}^{(t)} &= 1, \\
m_i^{(t)} &= v_i', \quad i = 1:K, & m_{K+1}^{(t)} &= 1, \\
v_i^{(t)} &= \widetilde{v}_i(1 - \gamma_i(\gamma_i + \alpha_i)), \quad i = 1:K & v_{K+1}^{(t)} &= 0,
\end{aligned}$$

and

$$\begin{aligned}
\rho^{(y)} &= 1 - \prod_{i=1}^{K+1}(1 - \rho_i^{(t)}) = 1 - \prod_{i=1}^{K}(1 - \Phi(\alpha_i))(1-1) = 1 \\
m^{(y)} &= \frac{1}{\sqrt{K+1}} \sum_{i=1}^{K+1} m_i \rho_i^{(t)} m_i^{(t)} \\
&= \frac{1}{\sqrt{K+1}} \left(m_{K+1} + \sum_{i=1}^{K} m_i \Phi(\alpha_i) v_i' \right) \\
(K+1)v^{(y)} &= -\rho^{(y)}(1-\rho^{(y)})(m^{(y)})^2 + \mathbf{v}^\top(\boldsymbol{\rho}^{(t)} \circ \mathbf{m}^{(t)} \circ \mathbf{m}^{(t)}) \\
&\quad + (\mathbf{m} \circ \mathbf{m} + \mathbf{v})^\top(\boldsymbol{\rho}^{(t)} \circ \boldsymbol{v}^{(t)}) \\
&\quad + (\mathbf{m} \circ \mathbf{m})^\top(\boldsymbol{\rho}^{(t)} \circ (1 - \boldsymbol{\rho}^{(t)}) \circ \mathbf{m}^{(t)} \circ \mathbf{m}^{(t)}) \\
&= v_{K+1} + \sum_{i=1}^{K}(m_i^2 + v_i)(1 - \Phi(\alpha_i))\Phi(\alpha_i)(v_i')^2 \\
&\quad + (m_i^2 + v_i)\Phi(\alpha_i)\widetilde{v}_i(1 - \gamma_i(\gamma_i + \alpha_i)) + v_i\Phi(\alpha_i)^2(v_i')^2.
\end{aligned} \tag{A.4}$$

Comparing Equations A.3 and A.4, we see the two methods are exactly equivalent after a linear combination step, with slab probability equal to 1 in the spike-and-slab variant. As such, any future hidden layers will behave identically. In the regression setting of PBP, the final transformation is solely a linear combination, so the final output is the same.

A.3 Additional Empirical Results

We include here additional results for RMSE and log-likelihood of different model configurations. Table A.1 and Tables A.2 and A.3 report the RMSE and

	1 Layer 10 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**3.787±0.344**	3.789±0.351
Combined Cycle Power Plant	**4.057±0.046**	4.068±0.042
Concrete Compression Strength	**6.221±0.222**	6.426±0.184
Energy Efficiency	2.100±0.082	**2.086±0.077**
Kin8nm	0.140±0.002	**0.137±0.003**
Naval Propulsion	**0.009±0.000**	**0.009±0.000**
Wine Quality Red	0.619±0.010	**0.617±0.009**
Yacht Hydrodynamics	1.769±0.197	**1.747±0.177**
	1 Layer 100 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**3.432±0.244**	3.437±0.255
Combined Cycle Power Plant	**4.157±0.025**	4.159±0.024
Concrete Compression Strength	**5.565±0.056**	5.574±0.028
Energy Efficiency	**1.879±0.063**	1.898±0.069
Kin8nm	**0.091±0.000**	**0.091±0.001**
Naval Propulsion	**0.004±0.000**	**0.004±0.000**
Wine Quality Red	0.630±0.014	**0.628±0.014**
Yacht Hydrodynamics	1.220±0.057	**1.176±0.054**

Table A.1: Mean and standard error of average test set RMSE of PBP and SSPBP, on eight datasets.

log-likelihood for the standard version of PBP and SSPBP, respectively. Similarly, Table A.4 and Tables A.5 and A.6 report the RMSE and log-likelihood of the "bias-free" versions of PBP and SSPBP, respectively.

	1 Layer 10 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**-2.810±0.140**	-2.823±0.141
Combined Cycle Power Plant	**-2.821±0.010**	-2.824±0.010
Concrete Compression Strength	**-3.253±0.039**	-3.284±0.032
Energy Efficiency	-2.179±0.053	**-2.169±0.047**
Kin8nm	0.549±0.017	**0.567±0.021**
Naval Propulsion	3.275±0.008	**3.276±0.005**
Wine Quality Red	-0.941±0.017	**-0.937±0.015**
Yacht Hydrodynamics	-2.022±0.083	**-2.012±0.070**
	1 Layer 100 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	-2.727±0.106	**-2.725±0.111**
Combined Cycle Power Plant	**-2.844±0.006**	-2.845±0.006
Concrete Compression Strength	**-3.136±0.011**	-3.138±0.006
Energy Efficiency	**-2.056±0.034**	-2.066±0.038
Kin8nm	0.968±0.003	**0.975±0.005**
Naval Propulsion	**3.949±0.005**	3.945±0.006
Wine Quality Red	-0.959±0.027	**-0.955±0.026**
Yacht Hydrodynamics	-1.751±0.017	**-1.739±0.011**

Table A.2: Mean and standard error of the test set log-likelihood of PBP and SSPBP, on eight datasets.

	1 Layer 50 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**-2.771±0.076**	-2.787±0.087
Combined Cycle Power Plant	**-2.834±0.009**	**-2.834±0.008**
Concrete Compression Strength	**-3.149±0.025**	-3.157±0.022
Energy Efficiency	-2.049±0.042	**-2.047±0.041**
Kin8nm	**0.901±0.008**	0.885±0.010
Naval Propulsion	**3.725±0.006**	3.717±0.007
Wine Quality Red	-1.002±0.006	**-1.001±0.006**
Yacht Hydrodynamics	**-1.767±0.027**	-1.775±0.031
	2 Layers 10 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	-2.576±0.073	**-2.535±0.076**
Combined Cycle Power Plant	**-2.828±0.017**	-2.830±0.016
Concrete Compression Strength	-3.218±0.028	**-3.201±0.029**
Energy Efficiency	**-1.802±0.025**	-1.878±0.047
Kin8nm	0.784±0.034	**0.799±0.015**
Naval Propulsion	**3.713±0.019**	3.686±0.005
Wine Quality Red	-1.009±0.025	**-1.003±0.016**
Yacht Hydrodynamics	**-1.714±0.054**	-1.749±0.032

Table A.3: Mean and standard error of the test set log-likelihood of PBP and SSPBP, on eight datasets.

	1 Layer 10 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	4.120±0.104	**4.061±0.149**
Combined Cycle Power Plant	**4.330±0.033**	4.335±0.026
Concrete Compression Strength	**8.240±0.081**	8.242±0.102
Energy Efficiency	2.060±0.059	**2.054±0.067**
Kin8nm	**0.165±0.001**	**0.165±0.001**
Naval Propulsion	**0.009±0.000**	**0.009±0.000**
Wine Quality Red	0.642±0.003	**0.636±0.004**
Yacht Hydrodynamics	**6.248±0.381**	6.275±0.353

	1 Layer 100 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	3.279±0.146	**3.260±0.124**
Combined Cycle Power Plant	4.208±0.049	**4.207±0.047**
Concrete Compression Strength	6.726±0.190	**6.718±0.183**
Energy Efficiency	**1.901±0.051**	**1.901±0.052**
Kin8nm	**0.156±0.002**	**0.156±0.002**
Naval Propulsion	**0.005±0.000**	**0.005±0.000**
Wine Quality Red	0.637±0.008	**0.635±0.007**
Yacht Hydrodynamics	7.233±0.486	**6.215±0.152**

Table A.4: Mean and standard error of average test set RMSE of the bias-free versions of PBP and SSPBP, on eight datasets.

	1 Layer 10 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	-2.912±0.051	**-2.891±0.057**
Combined Cycle Power Plant	**-2.885±0.008**	-2.886±0.006
Concrete Compression Strength	**-3.551±0.015**	-3.555±0.018
Energy Efficiency	-2.151±0.033	**-2.148±0.038**
Kin8nm	0.382±0.005	**0.395±0.003**[†]
Naval Propulsion	**3.251±0.008**	3.239±0.008
Wine Quality Red	-0.978±0.006	**-0.969±0.007**
Yacht Hydrodynamics	**-3.285±0.079**	-3.291±0.076
	1 Layer 100 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**-2.597±0.048**	-2.589±0.040
Combined Cycle Power Plant	-2.857±0.012	**-2.856±0.011**
Concrete Compression Strength	-3.327±0.029	**-3.326±0.028**
Energy Efficiency	**-2.067±0.026**	-2.068±0.027
Kin8nm	**0.441±0.014**	**0.441±0.013**
Naval Propulsion	3.885±0.006	**3.895±0.012**
Wine Quality Red	-0.969±0.013	**-0.965±0.013**
Yacht Hydrodynamics	-3.335±0.046	**-3.213±0.027**

Table A.5: Mean and standard error of the test set log-likelihood of bias-free versions of PBP and SSPBP, on eight datasets.
[†] Due to numerical issues, the five trials for this model were repeated with a different random seed.

	1 Layer 50 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	**-2.699±0.105**	-2.709±0.108
Combined Cycle Power Plant	-2.879±0.013	**-2.878±0.012**
Concrete Compression Strength	-3.385±0.019	**-3.373±0.026**
Energy Efficiency	-2.011±0.017	**-2.010±0.019**
Kin8nm	0.421±0.013	**0.424±0.012**
Naval Propulsion	**3.673±0.006**	3.658±0.006
Wine Quality Red	-0.944±0.023	**-0.939±0.022**
Yacht Hydrodynamics	**-3.264±0.055**	-3.290±0.059
	2 Layers 10 Nodes	
Dataset	**PBP**	**SSPBP**
Boston Housing	-2.944±0.191	**-2.916±0.146**
Combined Cycle Power Plant	-2.852±0.004	**-2.851±0.006**
Concrete Compression Strength	**-3.343±0.034**	-3.349±0.033‡
Energy Efficiency	-1.902±0.018	**-1.850±0.021**
Kin8nm	**0.655±0.009**	0.662±0.005†
Naval Propulsion	**3.772±0.029**	3.703±0.021
Wine Quality Red	**-0.974±0.022**	-0.986±0.025
Yacht Hydrodynamics	**-2.687±0.054**	-2.730±0.068

Table A.6: Mean and standard error of the test set log-likelihood of bias-free versions of PBP and SSPBP, on eight datasets.
† Due to numerical issues, the five trials for this model were repeated with a different random seed.
‡ Due to numerical issues, we report results from seven trails out of ten that yielded finite values for the test set log-likelihood.

Appendix B: Exploration of Gas Dataset

Figure B.2 shows the results of a study on the Gas dataset, following Papamakarios et al. (2017). We selected the Gas dataset in part because it is relatively low-dimensional with considerable structure and some approximately marginally normal covariates (see Figure B.1). Here, we compare the test log-likelihood performance of the MAF model under different modifications, focusing on whether the weighted-likelihood bootstrap (WLB) or nonparametric learning (NPL) Fong et al. (2019) approaches offer discernible benefit.

In all of the experiments shown, 10 models were trained using early stopping when, after 30 epochs, the average validation set log-likelihood failed to improve, resetting to the best parameters found. The model with the best average validation set log-likelihood was selected, and the test set log-likelihood is presented in Figure B.2 (mean $\pm$ standard error). The original MAF model's results are in black, WLB results are in blue, and NPL results are in red and purple (getting darker as the concentration parameter α increases from left to right). The red results use pseudo-inputs drawn from the remaining elements of the training set not otherwise used in training (simulating a "historical data" approach), and the purple results use pseudo-inputs drawn from a multivariate normal prior with mean and covariance set to that of the training subset. WLB and NPL results with a circle marker indicate that the base distribution is a multivariate normal with mean and covariance set by the weighted minibatch (in this case, the full training subset), while results with a triangle marker only have the identity matrix as the covariance.

Each of the four main groups of results uses the same subset of the Gas training set, with the specified number of examples (500, 1000, 5000, 10000). Within each group, there are eight versions of the model, indicated by the lines near the top of figure.

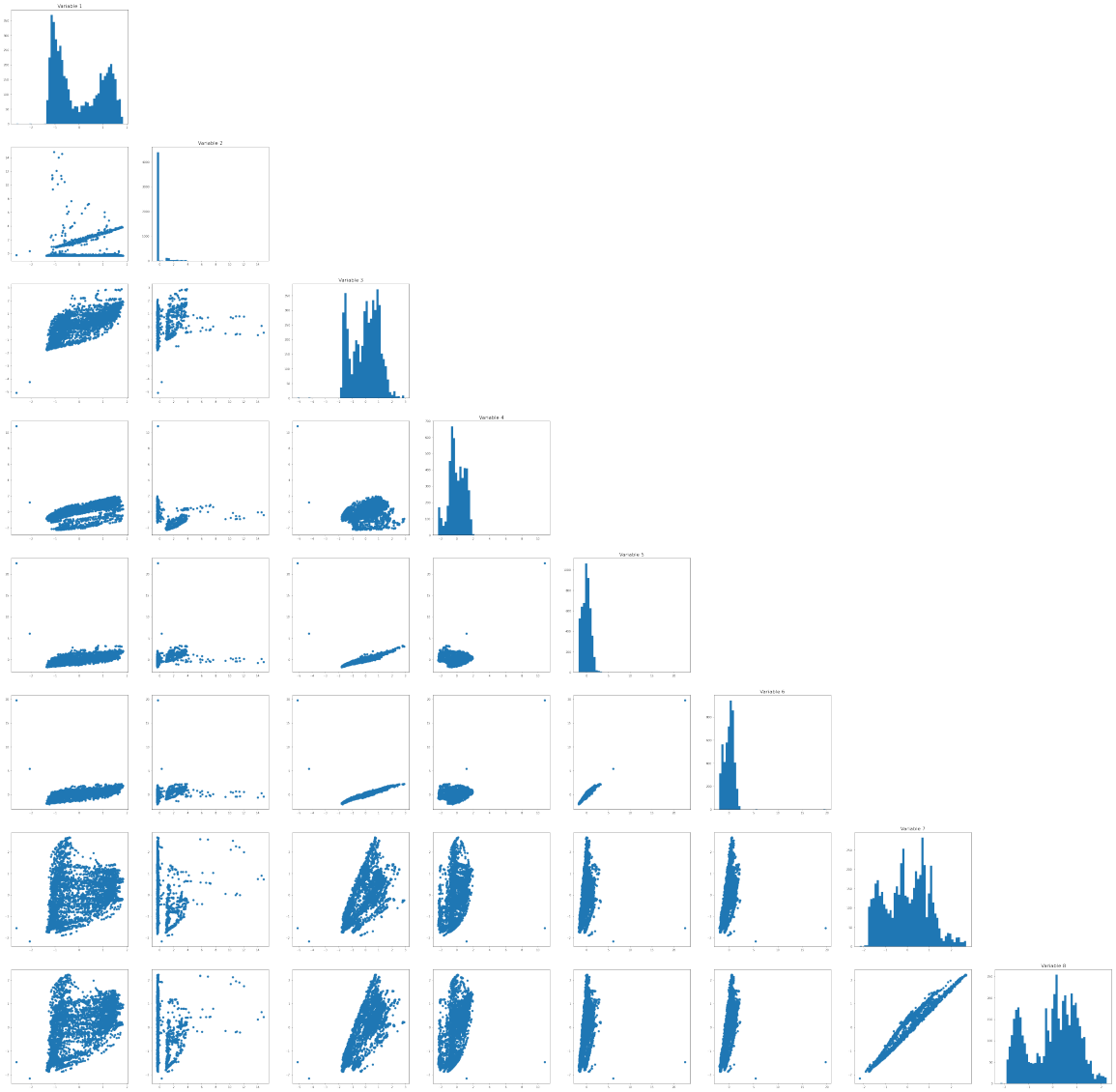

Figure B.1: Pairs plot of a sample of 5000 training set points of the Gas dataset (with histograms of each variable along the diagonal).

A black line in the "Init" row indicates that those models were preinitialized to an identity transformation, using 1000 epochs of 1000 examples from a uniform distribution on the bounding box of the training subset and a MSE loss between the examples and their preimage (unweighted). Our thinking (based on previous explorations not shown here) was that the MAF model seemed somewhat unstable, and that pretraining the flow to a sensible transformation may help that stability.

A black line in the "BN" row indicates that the MAF model uses batch normalization between each of the MADE layers (which was the case in the original MAF model). While batch normalization typically aids ML models, we experienced some numerical issues in MAF models and wanted to see if it was helpful in these settings.

Finally, a black line in the "Blocks=5" row indicates that the model uses five MADE blocks (or layers), following the original MAF paper, otherwise, it uses only one layer. This allows for a crude exploration of shallower and deep models.

Among the conclusions from this round of experiments, we learned:

- There are regimes where the Bayesian methods offer benefit against the original method, even with larger data on simpler models (e.g., all the mean-only WLB and NPL models with one block and N=10000).
- For the deeper models, some additional techniques seem necessary for good performance, whether preinitialization or batch normalization.
- For the NPL models, there was little relative distinction between the two priors for the pseudo-inputs, even though the "empirical" model allowed for information leak from the rest of the training set and the multivariate normal model did not.
- In sensible settings, the mean-only models seem prudent compared to the full covariance models.

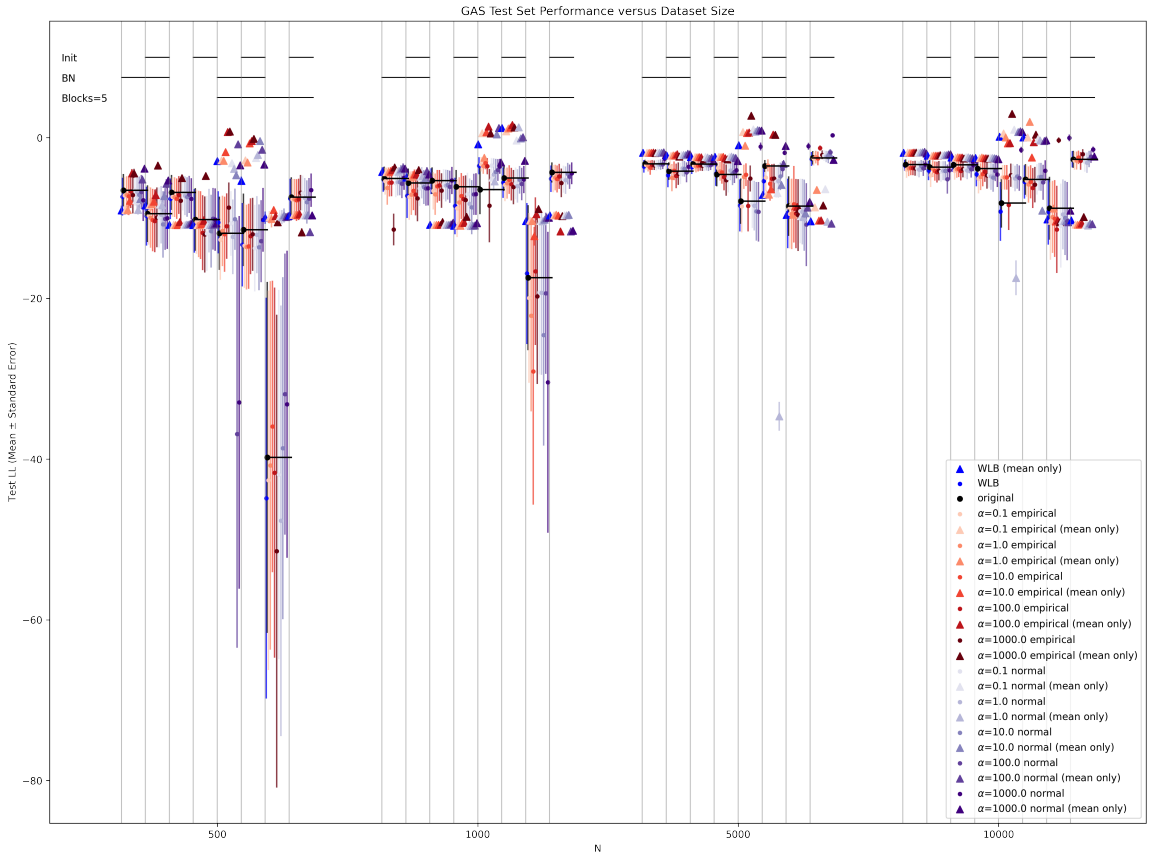

Figure B.2: Experimental results on the Gas dataset with the MAF model under many conditions. See main text (§B) for discussion. Best viewed in color.

Appendix C: Additional Background

This appendix includes brief elements of background not otherwise explicitly covered in the main text.

C.1 Distributional Distances and Divergences

The Kullback–Leibler (KL, Kullback and Leibler, 1951) divergence is an asymmetric means of comparing distributions:

$$KL(p\|q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx.$$

Maximum mean discrepancy (MMD, Gretton et al., 2012) is a kernel-based statistic to compare samples from two distributions; given a kernel function $k(x, y)$ and samples $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$, the MMD can be estimated using:

$$\mathrm{MMD}^2(X, Y) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k(x_i, x_j) + \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} k(y_i, y_j) - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} k(x_i, y_j).$$

C.2 Dirichlet Process

The Dirichlet process is a distribution over probability distributions. Given a concentration parameter α and base distribution H, samples $F \sim \mathrm{DP}(\alpha, H)$ have the property that, for any partition of the domain $\{S_i\}_{i=1}^{k}$,

$$(F(S_1), \ldots, F(S_k)) \sim \mathrm{Dir}(\alpha H(S_1), \ldots, \alpha H(S_k)).$$

This happens to correspond to a representation by a stick-breaking process with atoms $\{x_i\}_{i=1}^{\infty}$ as i.i.d. samples of H with

$$\beta_i \sim \text{Beta}(1, \alpha)$$
$$F = \sum_{i=1}^{\infty}(\beta_i \prod_{j=1}^{i-1}(1-\beta_j))\delta_{x_i}.$$

C.3 Additional Background for Graphs

C.3.1 Graph Statistics

Several graph statistics are helpful in comparing the structure of graphs. The *density* of a graph (with no repeated edges) is the number of edges in the graph divided by the total number of possible edges, giving an overall notion of how connected nodes in the graph are.

Node-level properties can also be aggregated to construct a graph-level statistic. $\mathcal{N}(v) = \{v_i | (v, v_i) \in E\}$ represents the neighbors of node v. Similarly, node v has degree $\deg(v) = |\mathcal{N}(v)|$. The graph's *degree distribution* is the probability distribution over all nodes' degrees.

The (local) clustering coefficient

$$C(v) = \frac{2\,|\{(v_i, v_j) \in E | v_i \in \mathcal{N}(v) \vee v_j \in \mathcal{N}(v)\}|}{\deg(v)(\deg(v) - 1}$$

measures how connected nodes in node v's neighborhood are, in comparison to its neighborhood being fully-connected. The *local clustering coefficient distribution*, then, provides a notion of the local connectedness aggregated over the whole graph, using each node's clustering coefficient.

Another useful graph statistic relies on the spectrum of the normalized graph Laplacian $L = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, where A is the adjacency matrix of the graph such that $A_{ij} = 1$ if and only if $(v_i, v_j) \in E$, D is the diagonal matrix of degrees with $D_{ii} = \deg(v_i)$, and I is the identity matrix. The eigenvalues of L provide a characteristic

of the graph that is useful in spectral graph theory. Of particular note is that the eigenvalues will all have values $0 \leq \lambda \leq 2$, making comparisons more straightforward. The *Laplacian spectrum distribution* aggregates these over multiple graphs.

C.3.2 Graphons

Graphons (Lovász, 2012) are symmetric functions $W : [0,1]^2 \to [0,1]$ that parameterize exchangeable simple graphs. From Orbanz and Roy (2014, Corollary III.6), a random simple (that is, undirected and without self-loops) graph G is vertex-exchangeable if and only if there is a graphon W such that its adjacency matrix A has:

$$A_{ij} \overset{\mathrm{d}}{=} \mathbb{1}\left[U_{\{i,j\}} < W(U_i, U_j)\right],$$

where $U_i, U_j, U_{\{i,j\}}$ are i.i.d. $U([0,1])$ random variables, independent of W. In other words, for every vertex v_i, we assign a representation U_i, with the probability of an edge between nodes v_i and v_j being given by the graphon $W(U_i, U_j)$. If W is constant everywhere, for example, we recover the Erdős et al. (1960) model for random graphs. Similarly, a stochastic block model can be recovered with a piecewise constant function based on a partition of the input domain $[0,1]$.

C.4 Weighted Likelihood Bootstrap

The Weighted Likelihood Bootstrap (Newton and Raftery, 1994) considers an alternate form of bootstrap sample. In this case, rather than resampling the data, each observation's contribution to the likelihood is given a random weight in the exponent, replacing the standard likelihood with $\prod_i p(y_i|\theta)^{w_i}$. The random weights are drawn from a Dirichlet distribution as $w \sim \mathrm{Dir}(1, \ldots, 1)$. Bootstrapped samples of θ can be obtained following Algorithm C.4.

Algorithm 5 Weighted Likelihood Bootstrap (from Lyddon et al. (2019)).

Observed samples are $y_{1:n}$
for $i = 1, \ldots, B$ **do**
  Draw random weights $g_i = (g_{i1}, \ldots, g_{in})$ with $n^{-1} g_i \sim \mathrm{Dir}(1, \ldots, 1)$
  $\theta^{(i)} = \arg\min_\theta \sum_{j=1}^{n} g_{ij} \log p(y_j|\theta)$
end for
Output $(\theta^{(1)}, \ldots, \theta^{(B)})$

C.5 Recurrent Neural Networks

Recurrent neural networks (RNNs, Rumelhart et al., 1986) are used in our model in Chapter 5 in the mechanism to update node embeddings, so we provide a brief introduction here. RNNs extend the basic structure of an FFNN to allow for sequential data. Assuming we have a sequence of vectors $x^{(t)} \in \mathbb{R}^p$, we could incorporate a state that persists between two elements of the sequence, such as:

$$h_\ell^{(t)} = \sigma_\ell(W_\ell h_{\ell-1}^{(t)} + U_\ell h_\ell^{(t-1)} + b_\ell),$$

taking for example $h_\ell^{(0)} = 0$, where U_ℓ is introduced as a weight matrix referring to any temporal relationship between elements of the sequence. This basic idea can has been extended in many ways, with some popular constructions being long short-term memory (LSTM, Hochreiter and Schmidhuber, 1997), or the gated recurrent unit (GRU, Cho et al., 2014). In these or other RNN approaches, we can still recover an efficient means of learning parameters through backpropagation by noting that the recurrent construction may be "unfolded" to form a computational sequence similar to that of a FFNN, only with parameters that are shared between the multiple places they appear.

Works Cited

David J Aldous. Representations for partially exchangeable arrays of random variables. *Journal of Multivariate Analysis*, 11(4):581–598, 1981.

Jincheng Bai, Qifan Song, and Guang Cheng. Adaptive variational bayesian inference for sparse deep neural network. *arXiv preprint arXiv:1910.04355*, 2019.

Jincheng Bai, Qifan Song, and Guang Cheng. Efficient variational inference for sparse deep learning with theoretical guarantee. *Advances in Neural Information Processing Systems*, 33:466–476, 2020.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Matt Benatan and Edward O Pyzer-Knapp. Practical considerations for probabilistic backpropagation. In *NeurIPS Workshop on Bayesian Deep Learning*, 2018.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Diana Cai, Trevor Campbell, and Tamara Broderick. Edge-exchangeable graphs and sparsity. *Advances in Neural Information Processing Systems*, 29, 2016.

François Caron and Emily B Fox. Sparse graphs using exchangeable random measures. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(5):1295–1366, 2017.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Adam D Cobb and Brian Jalaian. Scaling Hamiltonian Monte Carlo inference for Bayesian neural networks with symmetric splitting. In *Uncertainty in Artificial Intelligence*, 2021.

Harry Crane and Walter Dempsey. Edge exchangeable models for network data. *arXiv preprint arXiv:1603.04571*, 2016.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux-effortless Bayesian deep learning. *Advances in Neural Information Processing Systems*, 2021.

Dimah Dera, Ghulam Rasool, and Nidhal Bouaynaya. Extended variational inference for propagating uncertainty in convolutional neural networks. In *IEEE International Workshop on Machine Learning for Signal Processing*, 2019.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Shikai Fang, Shandian Zhe, Kuang-chih Lee, Kai Zhang, and Jennifer Neville. Online bayesian sparse learning with spike and slab priors. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 142–151. IEEE, 2020.

Edwin Fong, Simon Lyddon, and Chris Holmes. Scalable nonparametric sampling from multimodal posteriors with the posterior bootstrap. In *International Conference on Machine Learning*, pages 1952–1962. PMLR, 2019.

Jürgen Franke and Michael H Neumann. Bootstrapping neural networks. *Neural Computation*, 12(8):1929–1949, 2000.

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, 2016.

Jochen Gast and Stefan Roth. Lightweight probabilistic deep networks. In *Computer Vision and Pattern Recognition*, 2018.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pages 881–889. PMLR, 2015.

Elahe Ghalebi, Hamidreza Mahyar, Radu Grosu, Graham W Taylor, and Sinead A Williamson. A nonparametric bayesian model for sparse dynamic multigraphs. *arXiv preprint arXiv:1910.05098*, 2019.

Soumya Ghosh, Francesco Maria Delle Fave, and Jonathan S Yedidia. Assumed density filtering methods for learning Bayesian neural networks. In *AAAI Conference on Artificial Intelligence*, pages 1589–1595, 2016.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *International conference on machine learning*, pages 2434–2444. PMLR, 2019.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

Manuel Haußmann, Fred A Hamprecht, and Melih Kandemir. Sampling-free variational inference of Bayesian neural networks by variance backpropagation. In *Uncertainty in Artificial Intelligence*, 2020.

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, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

Douglas N Hoover. Relations on probability spaces and arrays of random variables. *Institute for Advanced Study*, 1979.

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.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, 2021. doi: 10.1109/TPAMI.2020.2992934.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018a.

Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. Stochastic expectation propagation. *Advances in neural information processing systems*, 28, 2015.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018b.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019.

László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.

Simon Lyddon, Stephen Walker, and Chris C Holmes. Nonparametric learning from Bayesian models with randomized objective functions. In *Advances in Neural Information Processing Systems*, 2018.

SP Lyddon, CC Holmes, and SG Walker. General bayesian updating and the loss-likelihood bootstrap. *Biometrika*, 106(2):465–478, 2019.

David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992a.

David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992b.

Thomas Peter Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL `www.graphlearning.io`.

Radford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

Michael A Newton and Adrian E Raftery. Approximate bayesian inference with the weighted likelihood bootstrap. *Journal of the Royal Statistical Society: Series B (Methodological)*, 56(1):3–26, 1994.

Leslie O'Bray, Max Horn, Bastian Rieck, and Karsten Borgwardt. Evaluation metrics for graph generative models: Problems, pitfalls, and practical solutions. *arXiv preprint arXiv:2106.01098*, 2021.

Lutz Oettershagen, Nils M Kriege, Christopher Morris, and Petra Mutzel. Temporal graph kernels for classifying dissemination processes. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pages 496–504. SIAM, 2020.

Manfred Opper. *On-line learning in neural networks*, chapter A Bayesian approach to on-line learning, page 363–378. Publications of the Newton Institute. Cambridge University Press, 1999.

Peter Orbanz and Daniel M Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):437–461, 2014.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018.

Evan Ott and Sinead Williamson. Nonparametric Posterior Normalizing Flows. submitted, 2022a.

Evan Ott and Sinead Williamson. Spike-and-Slab Probabilistic Backpropagation: When Smarter Approximations Make No Difference. In *I Can't Believe It's Not Better Workshop: Understanding Deep Learning Through Empirical Falsification*, 2022b. URL `https://openreview.net/forum?id=iYAdBHSA_Pt`.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

Wolfgang Roth and Franz Pernkopf. Variational inference in neural networks using an approximate closed-form objective. In *NeurIPS Workshop on Bayesian Deep Learning*, 2016.

Donald B Rubin. The bayesian bootstrap. *The annals of statistics*, pages 130–134, 1981.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Peter Schulam and Suchi Saria. Can you trust this prediction? auditing pointwise reliability after learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1022–1031. PMLR, 2019.

Minsuk Shin, Hyungjoo Cho, Hyun-seok Min, and Sungbin Lim. Neural bootstrapper. In *Advances in Neural Information Processing Systems*, 2021.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer, 2018.

Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning structured weight uncertainty in Bayesian neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2017.

Yan Sun, Qifan Song, and Faming Liang. Learning sparse deep neural networks with a spike-and-slab prior. *Statistics & Probability Letters*, 180:109246, 2022.

Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66 (2):145–164, 2013.

Esteban G Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

Luke Tierney and Joseph B Kadane. Accurate approximations for posterior moments and marginal densities. *Journal of the american statistical association*, 81(393):82–86, 1986.

Brian L Trippe and Richard E Turner. Conditional density estimation with Bayesian normalising flows. *arXiv preprint arXiv:1802.04908*, 2018.

Max Welling and Thomas N Kipf. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*, 2016.

Anqi Wu, Sebastian Nowozin, Edward Meeds, Richard E Turner, Jose Miguel Hernandez-Lobato, and Alexander L Gaunt. Deterministic variational inference for robust bayesian neural networks. *arXiv preprint arXiv:1810.03958*, 2018.

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1284–1293, 2019.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018.