Approximate Message Passing for Bayesian Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Bayesian methods have the ability to consider model uncertainty within a single framework and provide a powerful tool for decision-making. Bayesian neural networks (BNNs) hold great potential for better uncertainty quantification and data efficiency, making them promising candidates for more trustworthy AI in critical applications, and as backbones in data-constrained settings such as real-world reinforcement learning. However, current approaches often face limitations such as overconfidence, sensitivity to hyperparameters, and posterior collapse, highlighting the need for alternative approaches. In this paper, we introduce a novel method that leverages message passing (MP) to model the predictive posterior of BNNs as a factor graph. Unlike previous MP-based methods, our framework is the first to support convolutional neural networks (CNNs) while addressing the issue of double-counting training data, which has been a key source of overconfidence in prior work. Multiple open datasets are used to demonstrate the general applicability of the method and to illustrate its differences to existing inference methods.

1 Introduction

Deep learning models have achieved impressive results across various domains, including natural language processing [Vaswani et al., 2023], computer vision [Ravi et al., 2024], and autonomous systems [Bojarski et al., 2016]. Yet, they often produce overconfident but incorrect predictions, particularly in ambiguous or out-of-distribution scenarios. Without the ability to effectively quantify uncertainty, this can foster both overreliance and underreliance on models, as users stop trusting their outputs entirely [Zhang et al., 2024], and in high-stakes domains like healthcare or autonomous driving, its application can be dangerous [Henne et al., 2020]. To ensure safer deployment in these settings, models must not only predict outcomes but also express how uncertain they are about those predictions to allow for informed decision-making.

Bayesian neural networks (BNNs) offer a principled way to quantify uncertainty by capturing a posterior distribution over the model's weights, rather than relying on point estimates as in traditional neural networks (NNs). This allows BNNs to express epistemic uncertainty, the model's lack of knowledge about the underlying data distribution. Current methods for posterior approximation largely fall into two categories: sampling-based methods, such as Hamiltonian Monte Carlo (HMC), and deterministic approaches like variational inference (VI). While sampling methods are usually computationally expensive, VI has become increasingly scalable [Shen et al., 2024]. However, VI is not without limitations: It often struggles with overconfidence [Papamarkou et al., 2024], and it can struggle to break symmetry when multiple modes are close [Zhang et al., 2018]. Mean-field approaches, commonly used in VI, are prone to posterior collapse [Kurle et al., 2022, Coker et al., 2022]. Additionally, VI often requires complex hyperparameter tuning [Osawa et al., 2019], which complicates its practical deployment in real-world settings. These challenges motivate the need for alternative approaches that can address shortcomings of VI while maintaining its scalability.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

In contrast, message passing (MP) [Minka, 2001] is a probabilistic inference technique that suffers less from these problems. Belief propagation [Kschischang et al., 2001], the basis for many MP algorithms, integrates over variables of a joint density $p(\mathrm{x}_1, \ldots, \mathrm{x}_n)$ that factorize into a product of functions f_j on subsets of random variables $\mathrm{x}_1, \ldots, \mathrm{x}_n$. The corresponding factor graph is bipartite and connects these factors f_j with the variables they depend on. The following recursive equations yield a computationally efficient algorithm to compute all marginals $p(\mathrm{x}_i)$ for acyclic factor graphs:

$$p(\mathrm{x}) = \prod_{f \in N_\mathrm{x}} m_{f \to \mathrm{x}}(\mathrm{x}) \qquad \text{and} \qquad m_{f \to \mathrm{x}}(\mathrm{x}) \quad = \int f(N_f) \prod_{\mathrm{y} \in N_f \setminus \{\mathrm{x}\}} m_{\mathrm{y} \to f}(\mathrm{y})\, d(N_f \setminus \{\mathrm{x}\}),$$

where N_v denotes the neighborhood of vertex v and $m_{\mathrm{y} \to f}(\mathrm{y}) = \prod_{f' \in N_\mathrm{y} \setminus \{f\}} m_{f' \to \mathrm{y}}(\mathrm{y})$. Since exact messages are often intractable and factor graphs are rarely acyclic, belief propagation typically cannot be applied directly. Instead, messages $m_{f \to X}(\cdot)$ and marginals $p_X(\cdot)$ are typically approximated by some family of distributions that has few parameters (e.g., Gaussians). However, applying MP in practice presents two main challenges for practitioners: the need to derive (approximate) message equations when $m_{f \to \mathrm{x}}$ falls outside the approximating family, and the complexity of implementing MP compared to other methods.

Contributions Our contributions can be summarized as follows:

1. We propose a novel message-passing (MP) framework for BNNs and derive message equations for various factors, which can benefit factor graph modeling across domains.
2. We implement our method in Julia for both CPU and GPU, and demonstrate its general applicability to convolutional neural networks (CNNs) and multilayer perceptrons (MLPs) while avoiding the double-counting problem.
3. Having advantages for cases with few data (due to the Bayesian framework), we find that that our method is overall competitive with the SOTA baselines AdamW and IVON.
4. To the best of our knowledge, this is the first MP method to handle CNNs and to avoid double-counting training data, thereby preventing overconfidence and, eventually, posterior collapse.

2 Related Work

As the exact posterior is intractable for most practical NNs, approximate methods are essential for scalable BNNs. These methods generally fall into two categories: sampling-based approaches and those that approximate the posterior with parameterized distributions.

Markov Chain Monte Carlo (MCMC) methods attempt to draw representative samples from posterior distributions. Although methods such as Hamiltonian Monte Carlo are asymptotically exact, they become computationally prohibitive for large NNs due to their high-dimensional parameter spaces and complex energy landscapes [Coker et al., 2022]. An adaptation of Gibbs sampling has been scaled to MNIST, but on a very small network with only 8,180 parameters [Papamarkou, 2023]. Approximate sampling methods can be faster but still require a large number of samples, which complicates both training and inference. Although approaches like knowledge distillation [Korattikara et al., 2015] attempt to speed up inference, MCMC remains generally too inefficient for large-scale deep learning applications [Khan and Rue, 2024].

Variational Inference (VI) aims to approximate the intractable posterior distribution $p(\theta \mid \mathcal{D})$ by a variational posterior $q(\theta)$. The parameters of q are optimized using gradients with respect to an objective function, which is typically a generalized form of the reverse KL divergence $D_{\mathrm{KL}}\left[\, q(\theta) \,\|\, p(\theta \mid \mathcal{D}) \,\right]$. Early methods like [Graves, 2011] and Bayes By Backprop [Blundell et al., 2015] laid the foundation for applying VI to NNs, but suffer from slow convergence and severe underfitting, especially for large models or small dataset sizes [Osawa et al., 2019]. More recently, VOGN [Osawa et al., 2019] achieved Adam-like results on ImageNet LSVRC by applying a Gauss-Newton approximation to the Hessian matrix. IVON [Shen et al., 2024] improved upon VOGN by using cheaper Hessian approximations and training techniques like gradient clipping, achieving Adam-like performance on large-scale models such as GPT-2 while maintaining similar runtime costs. Despite recent advancements, VI continues to face challenges such as overconfidence, posterior collapse, and complex hyperparameter tuning (see introduction), motivating the exploration of alternative approaches [Zhang et al., 2018].

Message Passing (MP) for Neural Networks: MP is a general framework that unifies several algorithms [Kschischang et al., 2001, Minka, 2001], but its direct application to NNs has been limited. Expectation backpropagation (EBP) [Soudry et al., 2014] approximates the posterior of 3-layer MLPs by combining expectation propagation, an approximate MP algorithm, with gradient backpropagation. Similarly, probabilistic backpropagation (PBP) [Hernández-Lobato and Adams, 2015] combines belief propagation with gradient backpropagation and was found to produce better approximations than EBP [Ghosh et al., 2016]. However, PBP treats the data as new examples in each consecutive epoch (double-counting), which makes it prone to overconfidence. Furthermore, EBP and PBP were both only deployed on small datasets and rely on gradients instead of pure MP. In contrast, Lucibello et al. [2022] applied MP to larger architectures by modeling the posterior over NN weights as a factor graph, but faced posterior collapse to a point measure due to also double-counting data. Their experiments were mostly restricted to three-layer MLPs without biases and with binary weights. Our approach builds on this by introducing an MP framework for BNNs that avoids double-counting, scales to CNNs, and effectively supports continuous weights.

3 Theoretical Model

Our goal is to model the predictive posterior of a BNN as a factor graph and find a Gaussian approximation of the predictive posterior via belief propagation. Essentially, factor graphs are probabilistic modelling tools for approximating the marginals of joint distributions, provided that they factorize sufficiently. For a more comprehensive introduction on factor graphs and the sum-product algorithm, refer to Kschischang et al. [2001] BNNs, on the other hand, treat the parameters θ of a model $f_\theta : \mathbb{R}^d \longrightarrow \mathbb{R}^o$ as random variables with prior beliefs $p(\theta)$. Given a training dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^n$ of i.i.d. samples, a likelihood relationship $p(\boldsymbol{y} \mid \boldsymbol{x}, \theta) = p(\boldsymbol{y} \mid f_\theta(\boldsymbol{x}))$, and a new input sample $\boldsymbol{x}$, the goal is to approximate the predictive posterior distribution $p(\boldsymbol{y} \mid \boldsymbol{x}, \mathcal{D})$, which can be written as:

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \mathcal{D}) = \int p(\boldsymbol{y} \mid \boldsymbol{x}, \theta)\, p(\theta \mid \mathcal{D})\, d\theta. \tag{1}$$

This means that the density of the predictive posterior is the expected likelihood under the posterior distribution $p(\theta \mid \mathcal{D})$, which is proportional[1] to the product of the prior and dataset likelihood:

$$p(\theta \mid \mathcal{D}) \propto p(\theta) \prod_{i=1}^n p(\boldsymbol{y}_i \mid f_\theta(\boldsymbol{x})). \tag{2}$$

The integrand in Equation (1) exhibits a factorized structure that is well-suited to factor graph modeling. However, directly modelling the relationship $\boldsymbol{o} = f_\theta(\boldsymbol{x})$ with a single Dirac delta factor $\delta(\boldsymbol{o} - f_\theta(\boldsymbol{x}))$ does not yield feasible message equations. Therefore, we model the NN at scalar level by introducing intermediate latent variables connected by elementary Dirac delta factors. Figure 1 illustrates this construction for a simple MLP with independent weight matrices a priori. While the abstract factor graph in the figure uses vector variables for simplicity, we actually derive message equations where each vector component is treated as a separate scalar variable and all Dirac deltas depend only on scalar variables. For instance, if $d = 2$, the conceptual factor $\delta(\mathbf{o} - \mathbf{W}_2\mathbf{a})$ is replaced by four scalar factors: $\delta(\mathrm{p}_{jk} - \mathrm{w}_{jk}\mathrm{a}_k)$ for $j, k = 1, 2$, with intermediate variables p_{jk}, and two factors $\delta(\mathrm{o}_j - (\mathrm{p}_{j1} + \mathrm{p}_{j2}))$. By multiplying all factors in this expanded factor graph and integrating over intermediate results, we obtain a function in $\boldsymbol{x}, \boldsymbol{y}, \theta$ that is proportional to the integrand in Equation (1). Hence, the marginal of the unobserved target $\mathbf{y}$ is proportional to $p(\boldsymbol{y} \mid \boldsymbol{x}, \mathcal{D})$. When $\mathbf{y}$ connects to only one factor, its marginal matches its incoming message.

4 Approximations

Calculating a precise representation of the message to the target of an unseen input is intractable for large networks and datasets. The three primary reasons are, that a) nonlinearities and multiplication produce highly complex exact messages which are difficult to represent and propagate, b) the enormous size of the factor graph for large datasets, and c) the presence of various cycles in the graph. These challenges shape the message approximations as well as the design of our training and prediction procedures, which we address in the following sections.

[1] with a proportionality constant of $1/p(\mathcal{D})$

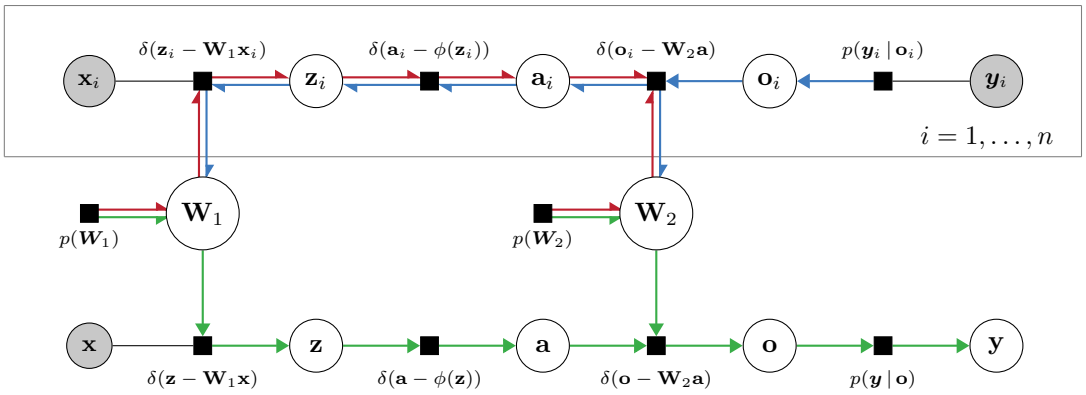


Figure 1: Conceptual vector-valued factor graph for a simple MLP. Each training example has its own "branch" (a copy of the network), while predictions for an unlabeled input $\boldsymbol{x}$ are computed on a separate prediction branch. All branches are connected by the shared model parameters. Grayed-out variables are conditioned on (observed). Colored arrows indicate the three iteration orders: a forward / backward pass on training examples, and a forward pass for prediction.

4.1 Approximating Messages via Gaussian Densities

To work around the highly complex exact messages, we approximate them with a parameterized class of functions. We desire this class to be closed under pointwise multiplication, as variable-to-factor messages are the product of incoming messages from other factors. We choose positive scalar multiples of one-dimensional Gaussian densities as our approximating family. Their closedness follows immediately from the exponential function's characteristic identity $\exp(x)\exp(y) = \exp(x+y)$ and the observation that for any $s_1, s_2 > 0$ and $\mu_1, \mu_2 \in \mathbb{R}$, the function $s_1(x-\mu_1)^2 + s_2(x-\mu_2)^2$ in x can be represented as $s(x-\mu)^2 + c$ for some $s > 0$ and $\mu, c \in \mathbb{R}$. The precise relation between two scaled Gaussian densities and its product can be neatly expressed with the help of the so-called natural (re-)parameterization. Given a Gaussian $\mathcal{N}(\mu, \sigma^2)$, we call $\rho = 1/\sigma^2$ the precision and $\tau = \mu/\sigma^2$ the precision-mean. Collectively, (τ, ρ) are the Gaussian's natural parameters, $\mathbb{G}(x; \tau, \rho) := \mathcal{N}(x; \mu, \sigma^2)$, $x \in \mathbb{R}$. For $\mu_1, \mu_2 \in \mathbb{R}$ and $\sigma_1, \sigma_2 > 0$ with corresponding natural parameters $\rho_i = 1/\sigma_i^2$ and $\tau_i = \mu_i\rho_i$, $i = 1, 2$, multiplying Gaussian densities simplifies to:

$$\mathcal{N}(\mu_1; \mu_2, \sigma_1^2 + \sigma_2^2) \cdot \mathbb{G}(x; \tau_1 + \tau_2, \rho_1 + \rho_2)$$

for all $x \in \mathbb{R}$. Thus, multiplying Gaussian densities simplifies to the pointwise addition of their natural parameters, aside from a multiplicative constant. Since we are only interested in the marginals, which are re-normalized, this constant does not affect the final result.

Next, we present our message approximations for three factor types, each representing a deterministic relationship between variables: (i) the sum of variables weighted by constants, (ii) the application of a nonlinearity, and (iii) the multiplication of two variables. As we model the factor graph on a scalar level, these three factors suffice to model complex modern network architectures such as ConvNeXt Liu et al. [2022][2]. In Appendix F, we provide a comprehensive table of message equations, including additional factors for modeling training labels.

Weighted Sum: The density transformation property of the Dirac delta allows us to compute the exact message without approximation. For the relationship $\mathrm{s} = \boldsymbol{c}^\intercal \mathbf{v}$ modeled by the factor $f := \delta(s - \boldsymbol{c}^\intercal \boldsymbol{v})$,

$$m_{f \to \mathrm{s}}(s) = \int \delta(s - \boldsymbol{c}^\intercal \boldsymbol{v}) \prod_{i=1}^{k} m_{\mathrm{v}_i \to f}(v_i)\, dv_1 \ldots v_k$$

is simply the density of $\boldsymbol{c}^\intercal \mathbf{v}$, where $\mathbf{v} \sim \prod_{i=1}^{k} m_{\mathrm{v}_i \to f}(v_i)$. If $m_{\mathrm{v}_i \to f}(v_i) = \mathcal{N}(v_i; \mu_i, \sigma_i^2)$ are Gaussian, then $\mathbf{v} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ and $m_{f \to \mathrm{s}}(s)$ becomes a scaled multivariate Gaussian:

$$m_{f \to \mathrm{s}}(s) = \mathcal{N}(s; \boldsymbol{c}^\intercal \boldsymbol{\mu}, (\boldsymbol{c}^2)^\intercal \boldsymbol{\sigma}^2).$$

The backward messages $m_{f \to \mathrm{v}_i}$ can be derived similarly without approximation.

[2]with the exception of layer normalization, which can be substituted by orthogonal initialization schemes Xiao et al. [2018] or specific hyperparameters of a corresponding normalized network Nguyen et al. [2023]

Nonlinearity: We model the application of a nonlinearity $\phi : \mathbb{R} \to \mathbb{R}$ as a factor $f := \delta(\mathrm{a} - \phi(\mathrm{z}))$. However, the forward and backward messages are problematic and require approximation–even for well-behaved, injective ϕ such as LeakyReLU$_\alpha$:

$$m_{\mathrm{a}\to f}(\mathrm{a}) = \mathrm{pdf}_{\phi(\mathrm{Z})}(\mathrm{a}) \;\text{ for }\; \mathrm{Z} \sim \mathcal{N}$$

$$m_{f\to \mathrm{z}}(\mathrm{z}) = \int \delta(\mathrm{a} - \phi(\mathrm{z})) \cdot m_{\mathrm{a}\to f}(\mathrm{a})\, d\mathrm{a} = m_{\mathrm{a}\to f}(\phi(z)) = \mathcal{N}(\phi(\mathrm{z}); \mu_\mathrm{a}, \sigma^2_\mathrm{a}).$$

For values of $\alpha \neq 1$, the forward message is non-Gaussian and the backward message does not even integrate to 1. For ReLU ($\alpha = 0$), it is clearly not even integrable. Instead, we use *moment matching* to fit a Gaussian approximation. Given any factor f and variable v, we can approximate the message $m_{f\to \mathrm{v}}$ directly with a Gaussian if the moments $m_k := \int \mathrm{v}^k m_{f\to\mathrm{v}}(\mathrm{v})\, d\mathrm{v}$ exist for $k = 0, 1, 2$ and can be computed efficiently via $m_{f\to\mathrm{v}}(\mathrm{v}) = \mathcal{N}(\mathrm{v}; m_1/m_0, m_2/m_0 - (m_1/m_0)^2)$. However, direct moment matching of the message is impossible for non-integrable messages or when the m_k are expensive to find. Instead, we can apply moment matching to the updated *marginal* of v. Let m_0, m_1, m_2 be the moments of the "true" marginal

$$m(\mathrm{v}) = \int f(\mathrm{v}, \mathrm{v}_1, ..., \mathrm{v}_k)\;\; d\mathrm{v}_1 ... d\mathrm{v}_k \;\cdot\; \prod\nolimits_i m_{g_i\to\mathrm{v}}(\mathrm{v}),$$

which is the product of the true message from f and the approximated messages from other factors g_i. Then we can approximate m with a Gaussian and obtain a message approximation

$$m_{f\to\mathrm{v}}(\mathrm{v}) := \mathcal{N}(\mathrm{v}; \mu_\mathrm{v}, \sigma^2_\mathrm{v}) / m_{\mathrm{v}\to f}(\mathrm{v})\,,$$

which approximates $m_{f\to\mathrm{v}}$ so that it changes v's marginal in the same way as the actual message.[3] Since $m_{\mathrm{v}\to f}(\mathrm{v})$ is a Gaussian density, we can compute $m_{f\to\mathrm{v}}(\mathrm{v})$ efficiently by applying Gaussian division in natural parameters, similar to Section 4.1. For LeakyReLU$_\alpha$, we found efficient direct and marginal approximations that are each applicable to both the forward and backward message when $\alpha \neq 0$. The marginal approximation remains applicable even for the ReLU case of $\alpha = 0$. We provide detailed derivations in Appendix B.2.

Product For the relationship $\mathrm{c} = \mathrm{ab}$, we employ variational MP as in Stern et al. [2009], in order to break the vast number of symmetries in the true posterior of a BNN. By combining the variational message equations for scalar products with the weighted sum, we can also construct efficient higher-order multiplication factors such as inner vector products, see Appendix F for detailed equations.

4.2 Training Procedure & Prediction

In pure belief propagation, the product of incoming messages for any variable equals its marginal under the true posterior. With our aforementioned approximations, we can reasonably expect to converge on a diagonal Gaussian $\check{q}$ that approximates one of the various permutation modes of the true posterior by aligning the first two moments of the marginal. This concept can be elegantly interpreted through the lens of relative entropy. As shown in A.2, among diagonal Gaussians $q(\theta) = q_1(\theta_1) \cdots q_k(\theta_k)$, the relative entropy from the true posterior to q is minimized for $\check{q}$:

$$\check{q} = \mathrm{argmin}_q\, D_{\mathrm{KL}}\left[\, p(\theta \,|\, \mathcal{D}) \,|\, q(\theta) \,\right]. \tag{3}$$

Another challenge in finding $\check{q}$ arises from cyclic dependencies. In acyclic factor graphs, each message depends only on previous messages from its subtree, allowing for exact propagation. However, our factor graph contains several cycles due to two primary reasons: (i) multiple training branches interacting with shared parameters across linear layers, and (ii) the scalar-level modeling of matrix-vector multiplication in architectures with more than one hidden layer. These loops create circular dependencies among messages. To address these challenges, we adopt loopy belief propagation, where belief propagation is performed iteratively until messages converge. While exact propagation works in acyclic graphs, convergence is then only guaranteed under certain conditions (e.g., Simon's condition [Ihler et al., 2005]) that are difficult to verify. Instead, we pass messages in an iteration order that largely avoids loops by alternating forward and backward passes similarly to deterministic NNs. Our message schedule is visualized in Figure 1.

[3] This is the central idea behind expectation propagation as defined in Minka [2001].

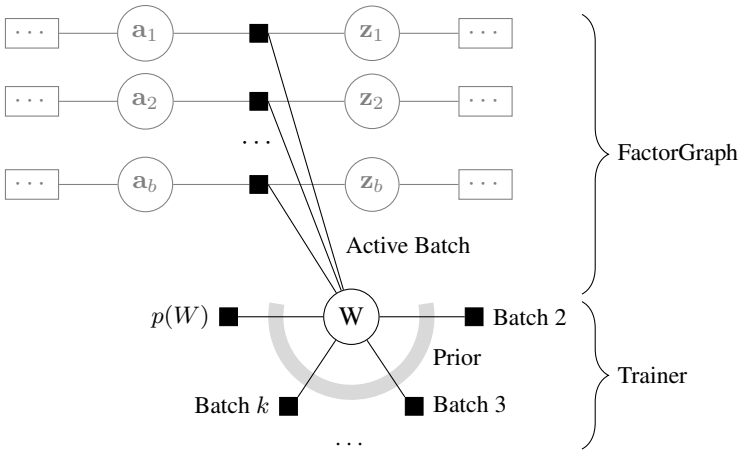


Figure 2: A full FactorGraph models all messages for one batch of training examples. To iterate, we only need one joint message summarizing the prior and all other examples. When switching to a new batch, we aggregate messages from the previous batch and store them in the Trainer.

Batching: As the forward and backward messages depend on each other, we must store them to compute message updates during message passing. Updating our messages in a sweeping "pass" over a branch and running backward passes immediately after the forward pass on the same branch, allows us to store many messages only temporarily, reducing memory requirements. This schedule also ensures efficient propagation of updated messages despite the presence of loops. However, some messages must still be retained permanently[4], leading to significant memory demand when storing them for all n training examples. To address this, we adopt a batching strategy: Instead of maintaining n training branches simultaneously, we update the factor graph using a batch (subset) of b examples at a time. The factor graph then models b messages to the weights W, while the messages to W from the remaining (inactive) examples are aggregated into batch-wise products and stored in a trainer object. Figure 2 illustrates this setup. When switching batches, we divide the marginals by the batch's old aggregate message and multiply the updated messages into the marginal, ensuring that data is not double-counted. Within each batch, we iterate through the examples and perform a forward and backward pass on each in sequence. After all examples have been processed once, we call it an "iteration". Depending on the training stage, we either repeat this process within the same batch or move to the next batch. As training progresses, we gradually increase the number of iterations per batch to allow for finer updates as the overall posterior comes closer to convergence.

Prediction: Ultimately, our goal is to compute the marginal of the unobserved target $\mathbf{y}$ for some unseen input $\mathbf{x}$. Since the prediction branch in Figure 1 introduces additional loops, obtaining an accurate approximation would require iterating over the entire factor graph, including the training branches. In NN terms, this translates to retraining the whole network for every test input. Instead, we pass messages only on the training branches in the batch-wise setup described above. At test time, messages from the training branches are propagated to the prediction branch, but not vice versa. Specifically, messages from the weights to the prediction branch are computed as the product of the prior and the incoming messages from the training branches. This can be interpreted as approximating the posterior over weights, $p(\theta \mid \mathcal{D})$, with a diagonal Gaussian $\check{q}(\theta)$ used as prior during inference.

4.3 Implementation

Scaling the approach to deep networks, the following challenges need to be addressed.

Factor Graph Implementation While batching effectively reduces memory requirements for large datasets, a direct implementation of a factor graph still scales poorly for deep networks. Explicitly modeling each scalar variable and factor as an instance is computationally expensive. To address this, we propose the following design optimizations: First, rather than modeling individual elements of the factor graph, we represent entire layers of the network. MP between layers is orchestrated by an outer training loop. Second, each layer instance operates across all training branches within the

[4]For example, the backward message of the linear layer is needed to compute the marginal of the inputs, which the forward message depends on.

active batch, removing the need to duplicate layers for each example. Third, factors are stateless functions, not objects. Each layer is responsible for computing its forward and backward messages by calling the required functions. In this design, layer instances maintain their own state, but MP and batching are managed in the outer loop. The stateless message equations are optimized for both performance and numerical stability. As a result, the number of layer instances scales linearly with network depth but remains constant regardless of layer size or batch size. This approach significantly reduces computational and memory overhead—our implementation is approximately 300x faster than a direct factor graph model in our tests. Additionally, we optimized our implementation for GPU execution by leveraging Julia's `CUDA.jl` and `Tullio.jl` libraries. Since much of the runtime is spent on linear algebra operations (within linear or convolutional layers), we built a reusable, GPU-compatible library for Gaussian multiplication. This design makes the implementation both scalable and extendable.

Numerical Stability Maintaining numerical stability in the MP process is critical, particularly as model size increases. Backward messages often exhibit near-infinite variances when individual weights have minimal impact on the likelihood. Therefore, we compute them directly in natural parameters, which also simplifies the equations. Special care is needed for LeakyReLU, as its messages can easily diverge. To mitigate this, we introduced guardrails: when normalization constants become too small, precision turns negative, or variance in forward messages increases, we revert to either $\mathbb{G}(0, 0)$ or use moment matching on messages instead of marginals (see Appendix F for details). Another trick is to periodically recompute the weight marginals from scratch to maintain accuracy. By leveraging the properties of Gaussians, we save memory by recomputing variable-to-factor messages as needed[5]. However, incremental updates to marginals can accumulates errors, so we perform a full recomputation once per batch iteration. Lastly, we apply light message damping through an exponential moving average to stabilize the training, but, importantly, only on the aggregated batch messages, not on the individual messages of the active batch.

Weight Priors A zero-centered diagonal Gaussian prior with variance σ_p^2 is a natural choice for the prior over weights. However, as in traditional deep learning, setting all means to zero prevents messages from breaking symmetry. To resolve this, we sample prior means according to spectral parametrization [Yang et al., 2024], which facilitates feature learning independent of the network width. Another challenge in prior choice is managing exploding variances. In a naive setup with $\sigma_p^2 = 1$, forward message variances grow exponentially with the network depth. To find a principled choice of σ_p^2, our initialization scheme is based on experimental data, see Appendix D.

5 Numerical Evaluation

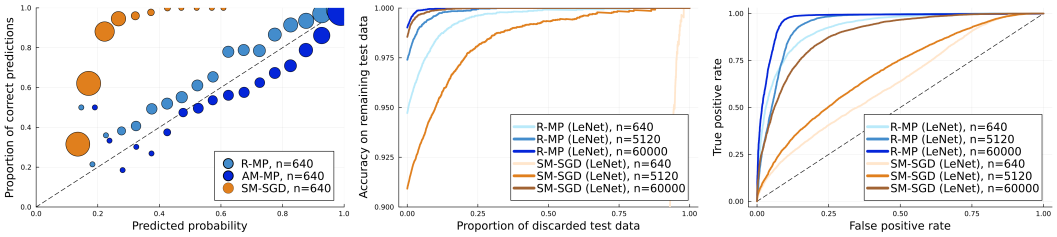
Experiment 1: Application on MNIST dataset. In our experiments on the MNIST dataset, we compare regression and classification-based versions of our message passing (MP) and SGD. Table 1 compares the test accuracy of MP and SGD for 3-layer MLPs and the LeNet-5 architecture [Lecun et al., 1998] over a range of training set sizes. We found that R-MP is generally more effective than AM-MP and that both consistently yield better accuracy than SGD, in particular for limited training data. For instance, our regression-based MP (R-MP) achieves 85.69% accuracy on the MLP with only 640 training samples, significantly outperforming softmax-based SGD's (SM-SGD) 58.85%. We also trained a 3-layer MLP of width 2,000 with 5.6 million parameters, which reached a test accuracy of 98.04%, whereas at width 256 the accuracy was 98.33%. Among related work on message passing, results for MNIST-sized datasets were only published by Lucibello et al. [2022]. Their method reached only 97.4% test accuracy and they published no metrics for evaluating their predictive uncertainty. For VI, Bayes By Backprop reported an accuracy of 98.18% for their Gaussian model and 98.64% for their mixture model, which are similar to the accuracy achieved by MP.

A key strength of our approach lies in the performance of its predictive uncertainty. Figure 3a shows that for a training dataset of size 640, counterintuitively, SGD is underconfident (ECE of 0.3695) whereas R-MP and AM-MP are both decently calibrated with an ECE of 0.0216 and 0.0251 respectively. All methods achieve good calibration when trained on the whole training data, with calibration errors of 0.0019 for SGD, 0.0014 for AM-MP, and 0.001 for R-MP. However, since most examples have high confidence levels, calibration becomes less informative at larger dataset sizes.

[5]Each layer stores factor-to-weight-variable messages and the marginal, which is an aggregate that is continuously updated as individual messages change.

	Num Data	80	160	320	640	1,280	2,560	5,120	10,240	60,000
MLP	**R-MP**	30.01	**61.79**	77.61	**85.69**	**88.95**	**91.72**	**94.85**	**96.25**	**98.33**
	AM-MP	**31.88**	61.08	**80.79**	85.50	87.92	91.56	94.08	95.72	98.21
	R-SGD	10.11	11.45	14.89	29.66	49.78	67.01	76.41	83.00	92.22
	SM-SGD	21.47	30.38	46.18	58.83	76.67	85.55	89.10	91.17	96.36
LeNet-5	**R-MP**	**27.75**	**25.58**	**38.02**	**94.72**	95.36	96.32	97.40	**98.12**	**99.02**
	AM-MP	17.32	10.42	10.28	93.48	**96.19**	**96.44**	**97.70**	98.05	98.95
	R-SGD	14.06	14.51	14.07	13.99	16.02	31.16	49.43	69.84	94.12
	SM-SGD	18.57	19.54	21.03	22.15	39.36	82.30	90.92	95.04	98.55

Table 1: Comparison of accuracies on MNIST (% correct). Our method (MP) consistently achieves better accuracy than SGD (Torch). Abbreviations: Regression (R), Argmax (AM), and Softmax (SM).



(a) Calibration at $n = 640$. (b) Relative calibration. (c) Out-of-distribution recognition.

Figure 3: Uncertainty metrics for models trained on MNIST.

Thus, we employ relative calibration curves to assess uncertainty further[6]. Figure 3b compares the relative calibration of R-MP and SM-SGD for LeNet-5. Overall, the R-MP predictions show excellent relative calibration with an area under the curve (AUC) of 0.9949, 0.9986, 0.9998 for 640, 5120, and 60 000 datapoints, whereas SM-SGD only achieved 0.4451, 0.9845, and 0.9995 respectively. Finally, we evaluated out-of-distribution (OOD) recognition by training a model on MNIST and then predicting on mixed examples from FashionMNIST and MNIST. Figure 3c shows the receiver operating characteristic (ROC) curve for detecting OOD samples by the entropy of their predicted class distribution. R-MP achieved an AUC of 0.9675 when trained on full MNIST and 0.9242 for $n = 640$, whereas SM-SGD only achieved 0.8872 even with the full training data.

A key strength of our approach lies in the performance of its predictive uncertainty. Figure 4a shows that for a training dataset of size 640, counterintuitively, SGD is underconfident (ECE of 0.3695) whereas R-MP and AM-MP are both decently calibrated with an ECE of 0.0216 and 0.0251 respectively. All methods achieve good calibration when trained on the whole training data, with calibration errors of 0.0019 for SGD, 0.0014 for AM-MP, and 0.001 for R-MP. However, since most examples have high confidence levels, calibration becomes less informative at larger dataset sizes. Thus, we employ relative calibration curves to assess uncertainty further6. Figure 4b compares the relative calibration of R-MP and SM-SGD for LeNet-5. Overall, the R-MP predictions show excellent relative calibration with an area under the curve (AUC) of 0.9949, 0.9986, 0.9998 for 640, 5120, and 60 000 datapoints, whereas SM-SGD only achieved 0.4451, 0.9845, and 0.9995 respectively. Finally, we evaluated out-of-distribution (OOD) recognition by training a model on MNIST and then predicting on mixed examples from FashionMNIST and MNIST. Figure 4c shows the receiver operating characteristic (ROC) curve for detecting OOD samples by the entropy of their predicted class distribution. R-MP achieved an AUC of 0.9675 when trained on full MNIST and 0.9242 for n=640, whereas SM-SGD only achieved 0.8872 even with the full training data.

Experiment 2: Application on CIFAR-10. To evaluate the applicability of our method on the CIFAR-10 dataset, we trained a 6 layer deep convolutional network with roughly 890k parameters on the full training dataset. As baseline methods we used the SOTA optimizers AdamW [Loshchilov and Hutter, 2017] and IVON [Shen et al., 2024] each with a cosine annealing learning rate schedule [Loshchilov and Hutter, 2016]. Across all methods, including ours, we trained for 25 epochs. In

[6]We order the test examples by their predicted max-class probability. For each uncertainty cutoff, we then plot the accuracy on the remaining (more certain) test set. The area under this curve is also reported under the name AUROC by Osawa et al. [2019].

	Acc. ↑	Top-5 Acc. ↑	NLL ↓	ECE ↓	Brier ↓	OOD-AUROC ↑
AdamW	**0.783**	**0.984**	1.736	0.046	0.38	0.792
IVON@mean	0.772	0.983	1.494	0.041	0.387	**0.819**
IVON	0.772	0.983	1.316	0.035	0.37	0.808
MP (Ours)	0.773	0.977	**0.997**	**0.029**	**0.361**	0.810

Table 2: Comparison of various validation statistics for a convolutional network of roughly 890k parameters trained on CIFAR-10. Out-of-distribution (OOD) detection was tested with SVHN. For IVON we used 100 samples for prediction at test time. IVON@mean are the results obtained from evaluating the model at the means of the learned distributions of the individual parameters.

Appendix C, we give extensive details on the network architecture and the experimental setup in general. Table 2 compares the performance of our method (MP) against AdamW and IVON across a variety of standard metrics. In general, we see that MP can compete with these two strong baselines. In the expected calibration error, our method even has a notable edge. The fact that the metrics are overall worse than what is reported by Shen et al. [2024] is probably due to a difference in architecture; Shen et al. only conduct experiments on ResNets equipped with filter response normalization [Singh and Krishnan, 2019]. Neither residual connections nor normalization layers are yet implemented in our factor graph library. Nevertheless, the potential of the approach becomes already visible.

Experiment 3: Further Evaluations on Tabular Benchmark Data. We use the UCI machine learning repository, cf. Dua and Graff [2017], for various regression tasks. The results, see Appendix E.1, show that our method is general applicable and effectively avoids overfitting.

6 Summary, Limitations & Future Work

Summary We presented a novel framework that advances message-passing (MP) for BNNs by modeling the predictive posterior as a factor graph. To the best of our knowledge, this is the first MP method to handle CNNs while avoiding double-counting training data, a limitation in previous MP approaches like Soudry et al. [2014], Hernández-Lobato and Adams [2015], Lucibello et al. [2022]. In our experiments on different datasets, our method proved to be competitive with the SOTA baselines AdamW and IVON, even showing an edge in terms of calibration.

Limitations Despite recent advances, VI methods like IVON remain ahead in scale and performance on larger datasets. Our approach's runtime and memory requirements scale linearly with model parameters and dataset size. While our inference at test time can keep up with IVON's sampling approach in terms of speed and memory requirements, training is up to two orders of magnitude slower and more GPU-memory intensive compared to training deterministic networks using PyTorch with optimizers like AdamW. The memory overhead stems from two key factors: First, each training example stores messages proportional to the model's parameter count, unlike AdamW's batch-level intermediate representations. Second, each parameter requires two 8-byte floating-point numbers, contrasting with more efficient 4-byte or smaller formats. Runtime inflation results from several performance bottlenecks: Our training schedule lacks parallel forward passes, our Tullio-based CUDA kernel generation misses memory-layout and GPU optimizations present in mature libraries like Torch, message equations involve complex computations beyond standard matrix multiplications, and we use Julia's default FP64 precision, which GPUs process less efficiently.

Future Work Regarding training efficiency, an altered message-update schedule with actual batched computations would significantly reduce training time. Implementing our library in CUDA C++ with efficiency in mind could also drastically cut down computational overhead. On the architectural front, we deem it likely that our approach can be extended to most modern deep learning architectures. Residual connections are straightforward to implement as they boil down to simple sum factors. For normalization layers at the scalar level, only a division factor is missing, which can be approximated by a "rotated" product factor. This would suffice to model ResNet-like architectures and more modern convolutional networks like ConvNeXt. For transformers, the last ingredient needed is an efficient softargmax factor. Given the division factor, only an exp factor is missing to model softargmax at the scalar level.

Reproducibility All code is available at https://github.com/neurips-submission-19866/submission.

References

Heart Failure Clinical Records. UCI Machine Learning Repository, 2020. DOI: https://doi.org/10.24432/C5Z89R.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks, 2015. URL https://arxiv.org/abs/1505.05424.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016. URL https://arxiv.org/abs/1604.07316.

Beau Coker, Wessel P. Bruinsma, David R. Burt, Weiwei Pan, and Finale Doshi-Velez. Wide mean-field bayesian neural networks ignore the data, 2022. URL https://arxiv.org/abs/2202.11670.

Paulo Cortez and Anbal Morais. Forest Fires. UCI Machine Learning Repository, 2007. DOI: https://doi.org/10.24432/C5D88D.

Paulo Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Wine Quality. UCI Machine Learning Repository, 2009. DOI: https://doi.org/10.24432/C56S3T.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux – effortless bayesian deep learning, 2022. URL https://arxiv.org/abs/2106.14806.

Dheeru Dua and Casey Graff. Uci machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Hadi Fanaee-T. Bike Sharing. UCI Machine Learning Repository, 2013. DOI: https://doi.org/10.24432/C5W894.

Soumya Ghosh, Francesco Delle Fave, and Jonathan Yedidia. Assumed density filtering methods for learning bayesian neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016. doi: 10.1609/aaai.v30i1.10296. URL https://ojs.aaai.org/index.php/AAAI/article/view/10296.

Alex Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf.

Maximilian Henne, Adrian Schwaiger, Karsten Roscher, and Gereon Weiss. Benchmarking uncertainty estimation methods for deep learning with safety-related metrics. In *SafeAI@ AAAI*, pages 83–90, 2020.

José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks, 2015. URL https://arxiv.org/abs/1502.05336.

Alexander Ihler, John III, and Alan Willsky. Loopy belief propagation: Convergence and effects of message errors. *Journal of Machine Learning Research*, 6:905–936, 05 2005.

Mohammad Emtiyaz Khan and Håvard Rue. The bayesian learning rule, 2024. URL https://arxiv.org/abs/2107.04562.

Anoop Korattikara, Vivek Rathod, Kevin Murphy, and Max Welling. Bayesian dark knowledge, 2015. URL https://arxiv.org/abs/1506.04416.

F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001. doi: 10.1109/18.910572.

Richard Kurle, Ralf Herbrich, Tim Januschowski, Yuyang Wang, and Jan Gasthaus. On the detrimental effect of invariances in the likelihood for variational inference, 2022. URL https://arxiv.org/abs/2209.07157.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022. URL https://arxiv.org/abs/2201.03545.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. URL http://arxiv.org/abs/1608.03983.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL http://arxiv.org/abs/1711.05101.

Carlo Lucibello, Fabrizio Pittorino, Gabriele Perugini, and Riccardo Zecchina. Deep learning via message passing algorithms based on belief propagation. *Machine Learning: Science and Technology*, 3(3):035005, jul 2022. doi: 10.1088/2632-2153/ac7d3b. URL https://dx.doi.org/10.1088/2632-2153/ac7d3b.

Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, page 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1558608001.

Warwick Nash, Tracy Sellers, Simon Talbot, Andrew Cawthorn, and Wes Ford. Abalone. UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C55C7W.

Khanh-Binh Nguyen, Jaehyuk Choi, and Joon-Sung Yang. Eunnet: Efficient un-normalized convolution layer for stable training of deep residual networks without batch normalization layer. *IEEE Access*, 11:76977–76988, 2023. doi: 10.1109/ACCESS.2023.3244072.

Cameron Nugent. California housing prices, 2017. URL https://www.kaggle.com/datasets/camnugent/california-housing-prices. Dataset derived from the 1990 U.S. Census, originally published by Pace, R. Kelley, and Ronald Barry in "Sparse Spatial Autoregressions", Statistics and Probability Letters, 33 (1997) 291-297.

Kazuki Osawa, Siddharth Swaroop, Anirudh Jain, Runa Eschenhagen, Richard E. Turner, Rio Yokota, and Mohammad Emtiyaz Khan. Practical deep learning with bayesian principles, 2019. URL https://arxiv.org/abs/1906.02506.

Theodore Papamarkou. Approximate blocked gibbs sampling for bayesian neural networks, 2023. URL https://arxiv.org/abs/2208.11389.

Theodore Papamarkou, Maria Skoularidou, Konstantina Palla, Laurence Aitchison, Julyan Arbel, David Dunson, Maurizio Filippone, Vincent Fortuin, Philipp Hennig, José Miguel Hernández-Lobato, Aliaksandr Hubin, Alexander Immer, Theofanis Karaletsos, Mohammad Emtiyaz Khan, Agustinus Kristiadi, Yingzhen Li, Stephan Mandt, Christopher Nemeth, Michael A. Osborne, Tim G. J. Rudner, David Rügamer, Yee Whye Teh, Max Welling, Andrew Gordon Wilson, and Ruqi Zhang. Position: Bayesian deep learning is needed in the age of large-scale ai, 2024. URL https://arxiv.org/abs/2402.00809.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos, 2024. URL https://arxiv.org/abs/2408.00714.

Yuesong Shen, Nico Daheim, Bai Cong, Peter Nickl, Gian Maria Marconi, Clement Bazan, Rio Yokota, Iryna Gurevych, Daniel Cremers, Mohammad Emtiyaz Khan, and Thomas Möllenhoff. Variational learning is effective for large deep networks, 2024. URL https://arxiv.org/abs/2402.17641.

Saurabh Singh and Shankar Krishnan. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. *CoRR*, abs/1911.09737, 2019. URL http://arxiv.org/abs/1911.09737.

Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 963–971, Cambridge, MA, USA, 2014. MIT Press.

David Stern, Ralf Herbrich, and Thore Graepel. Matchbox: Large scale bayesian recommendations. In *Proceedings of the 18th International World Wide Web Conference*, January 2009. URL https://www.microsoft.com/en-us/research/publication/matchbox-large-scale-bayesian-recommendations/.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks, 2018. URL https://arxiv.org/abs/1806.05393.

Greg Yang, James B. Simon, and Jeremy Bernstein. A spectral condition for feature learning, 2024. URL https://arxiv.org/abs/2310.17813.

I-Cheng Yeh. Real Estate Valuation. UCI Machine Learning Repository, 2018. DOI: https://doi.org/10.24432/C5J30W.

Cheng Zhang, Judith Butepage, Hedvig Kjellstrom, and Stephan Mandt. Advances in variational inference, 2018. URL https://arxiv.org/abs/1711.05597.

Zelun Tony Zhang, Sebastian S Feger, Lucas Dullenkopf, Rulu Liao, Lukas Süsslin, Yuanting Liu, and Andreas Butz. Beyond recommendations: From backward to forward ai support of pilots' decision-making process. *arXiv preprint arXiv:2406.08959*, 2024.

A Proof of Global Minimization Objective

A.1 Moment-Matched Gaussians Minimize Cross-Entropy

Consider a scalar density p and a Gaussian $q(\theta) = \mathcal{N}(\theta, \mu, \sigma)$. Then

$$\min H(p,q) = \min\left(\int p(\theta)\log\left(\frac{p(\theta)}{q(\theta)}\right)d\theta\right) = \min\left(\frac{1}{2\sigma^2}\int p(\theta)(\theta-\mu)^2\,d\theta + \frac{\log(2\pi\sigma^2)}{2}\right).$$

It is well known that expectations minimize the expected mean squared error. In other words, the integral is minimized by setting μ to the expectation of p and is then equal to the variance of p. The necessary condition of a local minimum then yields that σ^2 must be the variance of p.

A.2 Proof of Equation (3) Global Minimization Objective

Let p be an arbitrary probability density on $\mathbb{R}^k$ with marginals $p_i(\theta_i) := \int p(\theta)\,d(\theta \setminus \theta_i)$ and denote by $\mathcal{Q}$ the set of diagonal Gaussians. Then for every $q(\boldsymbol{\theta}) = \prod_{i=1}^k q_i(\theta_i) \in \mathcal{Q}$ we can write the relative entropy from p to q as

$$\begin{aligned} D_{\mathrm{KL}}[\,p\,||\,q\,] &= \int p(\boldsymbol{\theta})\log\left(\frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right)d\boldsymbol{\theta} = -\sum_{i=1}^k \int p(\boldsymbol{\theta})\log(q(\theta_i))d\boldsymbol{\theta} - H(p) \\ &= -\sum_{i=1}^k \int_{\theta_i}\log(q_i(\theta_i))\int_{\boldsymbol{\theta}\setminus\theta_i} p(\boldsymbol{\theta})d(\boldsymbol{\theta}\setminus\theta_i) - H(p) = \sum_{i=1}^k H(p_i,q_i) - H(p). \end{aligned}$$

This shows that $D_{\mathrm{KL}}[\,p\,||\,q\,]$ is minimized by independently minimizing the summands $H(p_i, q_i)$. In combination with A.1 this completes the proof.

B Derivations of Message Equations

B.1 ReLU

A common activation function is the Rectified Linear Unit ReLU : $\mathbb{R} \to \mathbb{R}, z \mapsto \max(0, z)$.

Forward Message: Since ReLU is not injective, we cannot apply the density transformation property of the Dirac delta to the forward message

$$m_{f\to \mathrm{a}}(\mathrm{a}) = \int_{\mathrm{z}\in\mathbb{R}} \delta(\mathrm{a} - \mathrm{ReLU}(\mathrm{z}))m_{\mathrm{z}\to f}(\mathrm{z})\,d\mathrm{z}.$$

In fact, the random variable ReLU(Z) with $\mathrm{Z} \sim m_{z\to f}$ does not even have a density. A positive amount of weight, namely $\Pr[Z \le 0]$, is mapped to 0. Therefore

$$m_{f\to \mathrm{a}}(0) = \lim_{t\to 0}\int_{\mathrm{z}\in\mathbb{R}} \mathcal{N}(\mathrm{ReLU}(\mathrm{z}); 0, t^2)m_{\mathrm{z}\to f}(\mathrm{z})\,d\mathrm{z} \ge \lim_{t\to 0}\mathcal{N}(0;0,t^2)\min_{\mathrm{z}\in[-1,0]} m_{\mathrm{z}\to f}(\mathrm{z}) = \infty.$$

Apart from 0, the forward message is well defined everywhere, and technically null sets do not matter under the integral. However, moment-matching $m_{\mathrm{z}\to f}$ while truncating at 0 does not seem reasonable as it completely ignores the weight of $m_{\mathrm{z}\to f}$ on $\mathbb{R}_{\le 0}$. Therefore, we refrain from moment-matching the forward message of ReLU.

As an alternative, we consider a marginal approximation. That means, we derive formulas for

$$m_k := \int_{\mathrm{a}\in\mathbb{R}} \mathrm{a}^k m_{\mathrm{a}\to f}(\mathrm{a})m_{f\to \mathrm{a}}(\mathrm{a})\,d\mathrm{a}, \quad k \in \{0,1,2\} \tag{4}$$

and then set

$$m_{f\to \mathrm{a}}(\mathrm{a}) := \mathcal{N}(\mathrm{a}; m_1/m_0, m_2/m_0 - (m_1/m_0)^2) \,/\, m_{\mathrm{a}\to f}(\mathrm{a}).$$

By changing the integration order, we obtain

$$\begin{aligned} m_k &= \int_{\mathrm{a}\in\mathbb{R}} \mathrm{a}^k m_{\mathrm{a}\to f}(\mathrm{a}) \int_{z\in\mathbb{R}} \delta(\mathrm{a} - \mathrm{ReLU}(z)) m_{z\to f}(z)\, dz\, d\mathrm{a} \\ &= \int_{z\in\mathbb{R}} m_{z\to f}(z) \int_{\mathrm{a}\in\mathbb{R}} \delta(\mathrm{a} - \mathrm{ReLU}(z)) \mathrm{a}^k m_{\mathrm{a}\to f}(\mathrm{a})\, d\mathrm{a}\, dz \\ &= \int_{z\in\mathbb{R}} m_{z\to f}(z) \mathrm{ReLU}^k(z) m_{\mathrm{a}\to f}(\mathrm{ReLU}(z))\, dz \end{aligned}$$

Note that we end up with a well-defined and finite integral. Similar integrals arise in later derivations. For this reason we encapsulate part of the analysis in basic building blocks.

Building Block 1. We can efficiently approximate integrals of the form

$$\int_0^\infty z^k \mathcal{N}(z; \mu_1, \sigma_1^2) \mathcal{N}(z; \mu_2, \sigma_2^2)\, dz$$

where $\mu_1, \mu_2 \in \mathbb{R}, \sigma_1, \sigma_2 > 0$ and $k = 0, 1, 2$.

Proof. By Section 4.1 the integral is equal to

$$\begin{aligned} S^+ &= \mathcal{N}(\mu_1; \mu_2, \sigma_1^2 + \sigma_2^2) \int_0^\infty z^k \mathcal{N}\left(z; \mu, \sigma^2\right)\, dz \\ &= \mathcal{N}(\mu_1; \mu_2, \sigma_1^2 + \sigma_2^2) \begin{cases} \mathbb{E}[\mathrm{ReLU}^k(\mathcal{N}(\mu, \sigma^2))] & \text{for } k = 1, 2 \\ \Pr[-Z \le 0] = \phi(\mu/\sigma) & \text{for } k = 0 \end{cases} \end{aligned}$$

where

$$\mu = \frac{\tau}{\rho}, \quad \sigma^2 = \frac{1}{\rho}, \quad \tau = \frac{\mu_1}{\sigma_1^2} + \frac{\mu_2}{\sigma_2^2} \quad \text{and} \quad \rho = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}.$$

□

This motivates the derivation of efficient formulas for the moments of an image of a Gaussian variable under ReLU.

Building Block 2. Let $\mathrm{Z} \sim \mathcal{N}(\mu, \sigma^2)$. The first two moments of $\mathrm{ReLU}(\mathrm{Z})$ are then given by

$$\mathbb{E}[\mathrm{ReLU}(\mathrm{Z})] = \sigma\varphi(x) + \mu\phi(x) \tag{5}$$

$$\mathbb{E}[\mathrm{ReLU}^2(\mathrm{Z})] = \sigma\mu\varphi(x) + (\sigma^2 + \mu^2)\phi(x), \tag{6}$$

where $x = \mu/\sigma$ and φ, ϕ denote the pdf and cdf of the standard normal distribution, respectively.

Proof. The basic idea is to apply $\int z e^{-z^2/2}\, dz = -e^{-z^2/2}$. Together with a productive zero, one obtains

$$\begin{aligned} \sqrt{2\pi}\sigma\mathbb{E}[\mathrm{ReLU}(\mathrm{Z})] &= \int_0^\infty z e^{-\frac{(z-\mu)^2}{2\sigma^2}}\, dz = \sigma^2 \int_0^\infty \frac{(z-\mu)}{\sigma^2} e^{-\frac{(z-\mu)^2}{2\sigma^2}}\, dz + \mu \int_0^\infty e^{-\frac{(z-\mu)^2}{2\sigma^2}}\, dz \\ &= \sigma^2 \left[-e^{-\frac{(z-\mu)^2}{2\sigma^2}}\right]_0^\infty + \sqrt{2\pi}\sigma\mu \Pr[Z \ge 0] \\ &= \sigma^2 e^{-\frac{\mu^2}{2\sigma^2}} + \sqrt{2\pi}\sigma\mu \Pr\left[\frac{-Z+\mu}{\sigma} \le \frac{\mu}{\sigma}\right] \\ &= \sqrt{2\pi}\sigma^2\varphi(x) + \sqrt{2\pi}\sigma\mu\phi(x). \end{aligned}$$

Rearranging yields the desired formula for the first moment. For the second moment, we need to complete the square and perform integration by parts:

$$\begin{aligned} &\mathbb{E}[\mathrm{ReLU}^2(\mathrm{Z})] = \frac{1}{\sqrt{2\pi}\sigma} \int_0^\infty z^2 e^{-\frac{(z-\mu)^2}{2\sigma^2}}\, dz \\ &= \frac{1}{\sqrt{2\pi}\sigma} \left(\sigma^2 \int_0^\infty (z-\mu)\frac{z-\mu}{\sigma^2} e^{-\frac{(z-\mu)^2}{2\sigma^2}}\, dz + 2\mu \int_0^\infty z e^{-\frac{(z-\mu)^2}{2\sigma^2}}\, dz - \mu^2 \int_0^\infty e^{-\frac{(z-\mu)^2}{2\sigma^2}}\, dz\right) \\ &= \frac{\sigma^2}{\sqrt{2\pi}\sigma} \left(\left[-(z-\mu)e^{-\frac{(z-\mu)^2}{2\sigma^2}}\right]_0^\infty + \int_0^\infty e^{-\frac{(z-\mu)^2}{2\sigma^2}}\right) + 2\mu\mathbb{E}[\mathrm{ReLU}(Z)] - \mu^2\phi(x) \\ &= -\sigma\mu\varphi(x) + \sigma^2\phi(x) + 2\mu\mathbb{E}[\mathrm{ReLU}(\mathrm{Z})] - \mu^2\phi(x) = \sigma\mu\varphi(x) + (\sigma^2 + \mu^2)\phi(x). \end{aligned}$$

□

Building Block 3. Integrals of the form

$$S^- := \int_{-\infty}^{0} z^k \mathcal{N}(z; \mu_1, \sigma_1^2)\mathcal{N}(0; \mu_2, \sigma_2^2)\, dz$$

where $\mu_1, \mu_2 \in \mathbb{R}, \sigma_1, \sigma_2 > 0$ and $k = 0, 1, 2$ can be efficiently approximated.

Proof. Employing the substitution $z = -t$ gives

$$\begin{aligned} S^- &= \mathcal{N}(0; \mu_2, \sigma_2^2) \int_0^{\infty} (-1)^k t^k \mathcal{N}(-t; \mu_1, \sigma_1^2)\, dt = (-1)^k \mathcal{N}(0; \mu_2, \sigma_2^2) \int_0^{\infty} t^k \mathcal{N}(t; -\mu_1, \sigma_1^2)\, dt \\ &= (-1)^k \mathcal{N}(0; \mu_2, \sigma_2^2) \begin{cases} \mathbb{E}[\text{ReLU}(\mathcal{N}(-\mu_1, \sigma_1^2))] & \text{for } k = 1, 2 \\ \Pr[-Z \geq 0] = \phi(-\mu_1/\sigma_1) & \text{for } k = 0. \end{cases} \end{aligned}$$

□

Now let $m_{z\to f}(z) = \mathcal{N}(z; \mu_z, \sigma_z^2), m_{\mathrm{a}\to f}(\mathrm{a}) = \mathcal{N}(\mathrm{a}; \mu_\mathrm{a}, \sigma_\mathrm{a}^2)$ and consider the decomposition

$$m_k = \underbrace{\int_0^{\infty} z^k \mathcal{N}(z; \mu_z, \sigma_z^2)\mathcal{N}(z; \mu_\mathrm{a}, \sigma_\mathrm{a}^2)\, dz}_{S^+} + \underbrace{\int_{-\infty}^{0} \text{ReLU}^k(z)\mathcal{N}(z; \mu_z, \sigma_z^2)\mathcal{N}(0; \mu_\mathrm{a}, \sigma_\mathrm{a}^2)\, dz}_{S^-}.$$

Note that S^+ falls under Building Block 1 for any $k = 0, 1, 2$. The other addend S^- is equal to 0 for $k = 1, 2$, and is handled by Building Block 3 for $k = 0$.

Backward Message: By definition of the Dirac delta, the backward message is equal to

$$m_{f\to z}(z) = \int_{\mathrm{a}\in\mathbb{R}} \delta(\mathrm{a} - \text{ReLU}(z)) m_{\mathrm{a}\to f}(\mathrm{a})\, d\mathrm{a} = m_{\mathrm{a}\to f}(\text{ReLU}(z))$$

which is, of course, not integrable, so it cannot be interpreted as a scaled density. Instead, we apply marginal approximation by deriving formulas for

$$m_k := \int_{z\in\mathbb{R}} z^k m_{z\to f}(z) m_{f\to z}(z)\, dz, \quad k \in \{0, 1, 2\}$$

and then setting

$$m_{f\to z}(z) := \mathcal{N}(z; m_1/m_0, m_2/m_0 - (m_1/m_0)^2) \,/\, m_{z\to f}(z).$$

To this end, let $m_{z\to f}(z) = \mathcal{N}(z; \mu_z, \sigma_z^2)$ and $m_{\mathrm{a}\to f}(\mathrm{a}) = \mathcal{N}(\mathrm{a}; \mu_\mathrm{a}, \sigma_\mathrm{a}^2)$. Then we have

$$m_k = \underbrace{\int_0^{\infty} z^k \mathcal{N}(z; \mu_z, \sigma_z^2)\mathcal{N}(z; \mu_\mathrm{a}, \sigma_a^2)\, dz}_{S^+} + \underbrace{\int_{-\infty}^{0} z^k \mathcal{N}(z; \mu_z, \sigma_z^2)\mathcal{N}(0; \mu_a, \sigma_a^2)\, dz}_{S^-}.$$

The two addends S^+ and S^- are handled by Building Block 1 and Building Block 3, respectively.

B.2 Leaky ReLU

Another common activation function is the Leaky Rectified Linear Unit

$$\text{LeakyReLU}_\alpha : \mathbb{R} \to \mathbb{R}, z \mapsto \begin{cases} z & \text{for } z \geq 0 \\ \alpha z & \text{for } z < 0. \end{cases}$$

It is parameterized by some $\alpha > 0$ that is typically small, such as $\alpha = 0.1$. In contrast to ReLU, it is injective (and even bijective). For this reason the forward and backward messages are both integrable and can be approximated by both direct and marginal moment matching. The notation is shown in Figure 4.

Figure 4: A deterministic factor corresponding to the $\mathrm{LeakyReLU}_\alpha$ activation function.

Forward Message: It is easy to show that the density of $\mathrm{LeakyReLU}_\alpha(\mathcal{N}(\mu_z, \sigma_z^2))$ is given by

$$p(a) = \mathcal{N}(\mathrm{LeakyReLU}_{1/\alpha}(a); \mu_z, \sigma_z^2) \begin{cases} 1 & \text{for } z \geq 0 \\ 1/\alpha & \text{for } z < 0 \end{cases}$$

which only has one discontinuity point, namely 0. In particular, it is continuous almost everywhere. So by the density transformation property of Dirac's delta, we have $m_{f\to a}(a) = p(a)$ for almost all a. Under the integral we can therefore replace $m_{f\to a}(a)$ by $p(a)$. This justifies that the moments of $m_{f\to a}$ are exactly the moments of $(\mathrm{LeakyReLU}_\alpha)_* \mathcal{N}(\mu_z, \sigma_z^2)$. Its expectation is equal to

$$\begin{aligned}
\mathbb{E}\left[\mathrm{LeakyReLU}_\alpha(\mathcal{N}(\mu_z, \sigma_z^2))\right] &= \int_{-\infty}^{0} \alpha z \mathcal{N}(z; \mu_z, \sigma_z^2)\, dz + \int_0^\infty z \mathcal{N}(z; \mu_z, \sigma_z^2)\, dz \\
&= -\alpha \int_0^\infty t \mathcal{N}(t; -\mu_z, \sigma_z^2)\, dt + \int_0^\infty z \mathcal{N}(z; \mu_z, \sigma_z^2)\, dz \\
&= -\alpha \mathbb{E}[\mathrm{ReLU}(\mathcal{N}(-\mu_z, \sigma_z^2))] + \mathbb{E}[\mathrm{ReLU}(Z)].
\end{aligned}$$

Both addends are handled by Building Block 2. Yet we can get more insight by further substitution:

$$\begin{aligned}
\mathbb{E}[\mathrm{LeakyReLU}_\alpha(Z)] &= -\alpha(\sigma_z \varphi(-\mu_z/\sigma_z) - \mu_z \phi(-\mu_z/\sigma_z)) + \sigma_z \varphi(\mu_z/\sigma_z) + \mu_z \phi(\mu_z/\sigma_z) \\
&= (1-\alpha)(\sigma_z \varphi(\mu_z/\sigma_z) + \mu_z \phi(\mu_z/\sigma_z)) + \alpha \mu_z \\
&= (1-\alpha)\mathbb{E}[\mathrm{ReLU}(Z)] + \alpha \mathbb{E}[Z].
\end{aligned}$$

In the second to last equation, we use the identities $\varphi(-x) = \varphi(x)$ and $\phi(-x) = 1 - \phi(x)$. As such, the mean of $\mathrm{LeakyReLU}_\alpha(Z)$ is a convex combination of the mean of $\mathrm{ReLU}(Z)$ and the mean of Z. The function $\mathrm{LeakyReLU}_1$ the identity, and its mean is accordingly the mean of Z. For $\alpha = 0$, we recover the mean of $\mathrm{ReLU}(Z)$.

The second moment of $\mathrm{LeakyReLU}_\alpha(Z)$ decomposes to

$$\begin{aligned}
\mathbb{E}[\mathrm{LeakyReLU}_\alpha^2(Z)] &= \int_{-\infty}^{0} \alpha^2 z^2 \mathcal{N}(z; \mu_z, \sigma_z^2)\, dz + \int_0^\infty z^2 \mathcal{N}(z; \mu_z, \sigma_z^2)\, dz \\
&= \alpha^2 \int_0^\infty z^2 \mathcal{N}(z; -\mu_z, \sigma_z^2)\, dz + \int_0^\infty z^2 \mathcal{N}(z; \mu_z, \sigma_z^2)\, dz \\
&= \alpha^2 \mathbb{E}[\mathrm{ReLU}^2(\mathcal{N}(-\mu_z, \sigma_z^2))] + \mathbb{E}[\mathrm{ReLU}^2(\mathcal{N}(\mu_z, \sigma_z^2))].
\end{aligned}$$

Again, both addends are covered by Building Block 2, so approximating the forward message via direct moment matching is feasible.

A marginal approximation can also be found. For all $k = 0, 1, 2$ we have

$$\begin{aligned}
&\int_{a\in\mathbb{R}} a^k m_{a\to f}(a) m_{f\to a}(a)\, da = \int_{a\in\mathbb{R}} a^k m_{a\to f}(a) p(a)\, da \\
&= \underbrace{\frac{1}{\alpha} \int_{-\infty}^{0} a^k \mathcal{N}(a; \mu_a, \sigma_a^2) \mathcal{N}(a/\alpha; \mu_z, \sigma_z^2)\, da}_{S^-} + \underbrace{\int_0^\infty a^k \mathcal{N}(a; \mu_a, \sigma_a^2) \mathcal{N}(a; \mu_z, \sigma_z^2)\, da}_{S^+}
\end{aligned}$$

The term S^+ is handled by Building Block 1. The term S^- is equal to

$$\begin{aligned}
S^- &= \int_{-\infty}^{0} a^k \mathcal{N}(a; \mu_a, \sigma_a^2) \mathcal{N}(a; \alpha\mu_z, (\alpha\sigma_z)^2)\, da \\
&= (-1)^k \int_0^\infty a^k \mathcal{N}(a; -\mu_a, \sigma_a^2) \mathcal{N}(a; -\alpha\mu_z, (\alpha\sigma_z)^2)\, da
\end{aligned}$$

and therefore also covered by Building Block 1.

Backward Message: By the sifting property of the Dirac delta, the backward message is equal to

$$m_{f\to z}(z) = \int_{a\in\mathbb{R}} \delta(a - \text{LeakyReLU}_\alpha(z)) m_{a\to f}(a)\, da = m_{a\to f}(\text{LeakyReLU}_\alpha(z)).$$

As opposed to ReLU, the backward message is integrable. That means, we can also apply direct moment matching: For all $k = 0, 1, 2$ we have

$$\begin{aligned} m_{f\to z}(z) &= \int_{-\infty}^{0} z^k \mathcal{N}(\alpha z; \mu_a, \sigma_a^2)\, dz + \int_0^\infty z^k \mathcal{N}(z; \mu_a, \sigma_a^2)\, dz \\ &= \frac{(-1)^k}{\alpha} \int_0^\infty z^k \mathcal{N}(z; -\mu_a/\alpha, (\sigma_a/\alpha)^2)\, dz + \int_0^\infty z^k \mathcal{N}(z; \mu_a, \sigma_a^2)\, dz \end{aligned}$$

For $k = 1$ or $k = 2$, the integrals fall under Building Block 2 again. If $k = 0$, then

$$m_{f\to z}(z) = \frac{(-1)^k}{\alpha}\phi(-\mu_a/\sigma_a) + \phi(\mu_a/\sigma_a).$$

Again, we can also find a marginal approximation as well. For all $k = 0, 1, 2$, we can write

$$\begin{aligned} &\int_{z\in\mathbb{R}} z^k m_{z\to f}(z) m_{f\to z}(z)\, dz \\ =&\int_{-\infty}^{0} z^k \mathcal{N}(z; \mu_z, \sigma_z^2)\mathcal{N}(\alpha z; \mu_a, \sigma_a^2)\, dz + \int_0^\infty z^k \mathcal{N}(z; \mu_z, \sigma_z^2)\mathcal{N}(z; \mu_a, \sigma_a^2)\, dz \\ =&\frac{(-1)^k}{\alpha} \int_0^\infty z^k \mathcal{N}(z; -\mu_z, \sigma_z^2)\mathcal{N}(z; -\mu_a/\alpha, (\sigma_a/\alpha)^2)\, dz + \int_0^\infty z^k \mathcal{N}(z; \mu_z, \sigma_z^2)\mathcal{N}(z; \mu_a, \sigma_a^2)\, dz \end{aligned}$$

Since both integrals are covered by Building Block 1 we have derived direct and marginal approximations of LeakyReLU messages using moment matching.

B.3 Softmax

We model the soft(arg)max training signal as depicted in Table 6. For the forward message on the prediction branch, we employ the so-called "probit approximation" [Daxberger et al., 2022]:

$$m_{f\to \mathrm{c}}(i) = \int \text{softmax}(\mathbf{a})_i \mathcal{N}(\mathbf{a}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)\, d\mathbf{a} \approx \text{softmax}(\boldsymbol{t})_i,$$

where $t_j = \mu_j/(1 + \frac{\pi}{8}\sigma_j^2),\ j = 1, \ldots, d$. For the backward message on a training branch, to say a_d, we use marginal approximation. We hence need to compute the moments m_0, m_1, m_2 of the marginal of a_d via:

$$\begin{aligned} m_k &= \int \mathrm{a}_d^k\, \text{softmax}(\mathbf{a})_\mathrm{c}\, \mathcal{N}(\mathbf{a}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)\, d\mathbf{a} \\ &= \int_{\mathrm{a}_d} \mathrm{a}_d^k\, \mathcal{N}(\mathrm{a}_d; \mu_d, \sigma_d^2) \int_{\mathbf{a}\setminus\mathrm{a}_d} \text{softmax}(\mathbf{a})_i \prod_{j\neq i} \mathcal{N}(\mathrm{a}_j; \mu_j, \sigma_j^2)\, d(\mathbf{a}\setminus \mathrm{a}_d) d\mathrm{a}_d. \end{aligned}$$

We can reduce the inner integral to the probit approximation by regarding the point distribution δ_{a_d} as the limit of a Gaussian with vanishing variance:

$$\begin{aligned} &\int_{\mathbf{a}\setminus\mathrm{a}_d} \text{softmax}(\mathbf{a})_\mathrm{c} \prod_{j\neq d} \mathcal{N}(\mathrm{a}_j; \mu_j, \sigma_j^2)\, d(\mathbf{a}\setminus \mathrm{a}_d) \\ =&\int_{\mathbf{a}\setminus\mathrm{a}_d} \int_{\tilde{a}_d} \delta(\tilde{a}_d - a_d)\, \text{softmax}(a_1, \ldots, a_{d-1}, \tilde{a}_d) \prod_{j\neq d} \mathcal{N}(a_j; \mu_j, \sigma_j^2)\, d\tilde{a}_d\, d(\mathbf{a}\setminus \mathrm{a}_d) \\ =&\int_{\tilde{\mathbf{a}}\setminus\tilde{\mathrm{a}}_d} \lim_{\sigma\to 0} \int_{\tilde{a}_i} \text{softmax}(\tilde{\mathbf{a}})_\mathrm{c}\, \mathcal{N}(\tilde{a}_d; a_d, \sigma^2) \prod_{j\neq d} \mathcal{N}(\tilde{a}_j; \mu_j, \sigma_j^2)\, d\tilde{a}_i\, d\tilde{\mathbf{a}}_i \end{aligned}$$

By Lebesgue's dominated convergence theorem we obtain equality to

$$\lim_{\sigma\to 0}\int_{\tilde{\mathbf{a}}} \text{softmax}(\tilde{\mathbf{a}})_c \mathcal{N}(\tilde{a}_d; a_d, \sigma^2)\prod_{j\neq i}\mathcal{N}(\tilde{a}_j;\mu_j,\sigma_j^2)\,d\tilde{\mathbf{a}}$$

$$\approx \lim_{\sigma\to 0}\text{softmax}(\boldsymbol{t})_i = \text{softmax}(t_1,\ldots,t_{d-1},\mathrm{a}_d) \quad \text{where} \quad t_j = \begin{cases}\mu_j/(1+\frac{\pi}{8}\sigma_j^2) & \text{for } j\neq d\\ \mathrm{a}_d/(1+\frac{\pi}{8}\sigma^2) & \text{for } j=d.\end{cases}$$

Hence, we can approximate m_k by one-dimensional numerical integration of

$$m_k \approx \int_{\mathrm{a}_d} \mathrm{a}_d^k\,\mathcal{N}(\mathrm{a}_d;\mu_d,\sigma_d^2)\,\text{softmax}(t_1,\ldots,t_{d-1},\mathrm{a}_d)\,d\mathrm{a}_d.$$

C Experimental Setup

Synthetic Data - Depth Scaling: We generated a dataset of 200 points by randomly sampling x values from the range $[0, 2]$. The true data-generating function was

$$f(x) = 0.5x + 0.2\sin(2\pi\cdot x) + 0.3\sin(4\pi\cdot x).$$

The corresponding y values were sampled by adding Gaussian noise: $f(x) + \mathcal{N}(0, 0.05^2)$. For the architecture, we used a three-layer NN with the structure:

$$[\text{Linear}(1,16), \text{LeakyReLU}(0.1), \text{Linear}(16,16), \text{LeakyReLU}(0.1), \text{Linear}(16,1)].$$

A four-layer network has one additional $[\text{Linear}(16,16), \text{LeakyReLU}(0.1)]$ block in the middle, and a five-layer network has two additional blocks. For the regression noise hyperparameter, we used the true noise $\beta^2 = 0.05^2$. The models were trained for 500 iterations over one batch (as all data was processed in a single active batch).

Synthetic Data - Uncertainty Evaluation: The same data-generation process was used as in the depth-scaling experiment, but this time, x values were drawn from the range $[-0.5, 0.5]$. The network architecture remained the same as the three-layer network, but the width of the layers was increased to 32. We trained 100 networks with different random seeds on the same dataset. We define a p-credible interval for $0 \leq p \leq 1$ as:

$$[\text{cdf}^{-1}(0.5 - \frac{p}{2}), \text{cdf}^{-1}(0.5 + \frac{p}{2})].$$

For each credible interval mass p (ranging from 0 to 1 in steps of 0.01), we measured how many of the p-credible intervals (across the 100 posterior approximations) covered the true data-generating function. This evaluation was done at each possible x value (ranging from -20 to 20 in steps of 0.05), generating a coverage rate for each combination of p and x. For each p, we then computed the median for $x > 10$ and the median for $x < -10$. If we correlate the p values with the medians, we found that for the median obtained from positive x values the correlation was 0.96, for negative x it was 0.99, and for the combined set of medians it was 0.9.

CIFAR-10: For our CIFAR-10 experiments, we used the default train-test split and trained the following feed-forward network:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.model = nn.Sequential(
            # Block 1
            nn.Conv2d(3, 32, 3, padding=0),
            nn.LeakyReLU(0.1),
            nn.Conv2d(32, 32, 3, padding=0),
            nn.LeakyReLU(0.1),
            nn.MaxPool2d(2),
            # Block 2
            nn.Conv2d(32, 64, 3, padding=0),
            nn.LeakyReLU(0.1),
```

```
            nn.Conv2d(64, 64, 3, padding=0),
            nn.LeakyReLU(0.1),
            nn.MaxPool2d(2),
            # Head
            nn.Flatten(),
            nn.Linear(64 * 5 * 5, 512),
            nn.LeakyReLU(0.1),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        return self.model(x)
```

In the case of AdamW and IVON we trained with a cross-entropy loss on the softargmax of the network output. For our message passing method we used our argmax factor as a training signal instead of softargmax, see Appendix F. The reason is that for softargmax we only have message approximations relying on rather expensive numerical integration. In our library this factor graph can be constructed via

```
    fg = create_factor_graph([
                size(d.X_train)[1:end-1], # (3, 32, 32)
                # First Block
                (:Conv, 32, 3, 0), # (32, 30, 30)
                (:LeakyReLU, 0.1),
                (:Conv, 32, 3, 0), # (32, 28, 28)
                (:LeakyReLU, 0.1),
                (:MaxPool, 2), # (32, 14, 14)
                # Second Block
                (:Conv, 64, 3, 0), # (64, 12, 12)
                (:LeakyReLU, 0.1),
                (:Conv, 64, 3, 0), # (64, 10, 10)
                (:LeakyReLU, 0.1),
                (:MaxPool, 2), # (64, 5, 5)
                # Head
                (:Flatten,), # (64*5*5 = 1600)
                (:Linear, 512), # (512)
                (:LeakyReLU, 0.1),
                (:Linear, 10), # (10)
                (:Argmax, true)
            ], batch_size)
```

For all methods we used a batch size of 128 and trained for 25 epochs with a cosine annealing learning rate schedule. Concerning hyperparameters: For AdamW we found the standard parameters of $\text{lr} = 10^{-3}, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ and $\delta = 10^{-4}$ to work best. For IVON we followed the practical guidelines given in the Appendix of Shen et al. [2024].

To measure calibration, we used 20 bins that were split to minimize within-bin variance. For OOD recognition, we predicted the class of the test examples in CIFAR-10 (in-distribution) and SVHN (OOD) and computed the entropy over softmax probabilities for each example. We then sort them by negative entropy and test the true positive and false positive rates for each possible (binary) decision threshold. The area under this ROC curve is computed in the same way as for relative calibration.

D Prior Analysis

The strength of the prior determines the amount of data needed to obtain a useful posterior that fits the data. Our goal is to draw prior means and set prior variances so that the computed variances of all messages are on the order of $\mathcal{O}(1)$ regardless of network width and depth. It is not entirely clear if this would be a desirable property; after all, adding more layers also makes the network more expressive and more easily able to model functions with very high or low values. However, if we let the predictive prior grow unrestricted, it will grow exponentially, leading to numerical issues.

In the following, we analyze the predictive prior under simplifying assumptions to derive a prior initialization that avoids exponential variance explosion. While we fail to achieve this goal, our current prior variances are still informed by this analysis.

In the following, we assume that the network inputs are random variables. Then, the parameters of messages also become random variables, as they are derived from the inputs according to the message equations. Our goal is to keep the expected value of the variance parameter of the outgoing message at a constant size. We also assume that the means of the prior are sampled according to spectral initialization, as described in Section 4.3.

FirstGaussianLinearLayer - Input is a Constant

Each linear layer transforms some d_1-dimensional input $\mathbf{x}$ to some d_2-dimensional output $\mathbf{y}$ according to $\mathbf{y} = W\mathbf{x} + \mathbf{b}$. In the first layer, $\mathbf{x}$ is the input data. For this analysis, we assume each element x_i to be drawn independently from $x_i \sim \mathcal{N}(0, 1)$. Let $\mathbf{x}$ be a d_1-dimensional input vector, $\mathbf{m}_w$ be the prior messages from one column of W, and $z = \mathbf{w}'\mathbf{x}$ be the vector product before adding the bias.

During initialization of the weight prior, we draw the prior means using spectral parametrization and set the prior variances to a constant:

$$m_{w_i} = \mathcal{N}(\mu_{w_i}, \sigma_w^2) \text{ with } \mu_{w_i} \sim \mathcal{N}(0, l^2),$$
$$l = \frac{1}{\sqrt{k}} \cdot \min(1, \sqrt{\frac{d_2}{d_1}}).$$

By applying the message equations, we then approximate the forward message to the output with a normal distribution

$$m_z = \mathcal{N}(\mu_z, \sigma_z^2).$$

Because σ_z^2 depends on the random variables x_i, it is also a random variable that follows a scaled chi-squared distribution

$$\sigma_z^2 = \sum_{i=1}^{d_1} x_i^2 \cdot \sigma_w^2$$
$$\sigma_z^2 \sim \chi_{d_1}^2 \cdot \sigma_w^2$$

and its expected value is

$$\mathbb{E}[\sigma_z^2] = d_1 \cdot \sigma_w^2.$$

We conclude that we can control the magnitude of the variance parameter by choosing $\mathbb{E}[\sigma_z^2]$ and setting $\sigma_w^2 = \frac{\mathbb{E}[\sigma_z^2]}{d_1}$.

GaussianLinearLayer - Input is a Variable

In subsequent linear layers, the input $\mathbf{x}$ is not observed and we receive an approximate forward message that consists of independent normal distributions

$$m_{x_i} = \mathcal{N}(\mu_{x_i}, \sigma_{x_i}^2).$$

Following the message equations, the outgoing forward message to z then has a variance

$$\sigma_z^2 = \sum_{i=1}^{d_1} (\sigma_{x_i}^2 + \mu_{x_i}^2) \cdot (\sigma_w^2 + \mu_{w_i}^2) - (\mu_{x_i}^2 * \mu_{w_i}^2)$$
$$= \sum_{i=1}^{d_1} \underbrace{\sigma_{x_i}^2 \cdot \sigma_w^2}_{\text{I}} + \underbrace{\sigma_{x_i}^2 \cdot \mu_{w_i}^2}_{\text{II}} + \underbrace{\mu_{x_i}^2 \cdot \sigma_w^2}_{\text{III}}$$

The layer's prior variance σ_w^2 is a constant, whereas all other elements are random variables according to our assumptions. To make further analysis tractable, we also have to assume that the variances

$\sigma^2_{x_i}$ of the incoming forward messages are identical constants for all i, not random variables. We furthermore assume that the means are drawn i.i.d. from:

$$\mu_{w_i} \sim \mathcal{N}(0, l^2)$$
$$\mu_{x_i} \sim \mathcal{N}(\mu_{\mu_x}, \sigma^2_{\mu_x}).$$

The random variable σ^2_z then follows a generalized chi-squared distribution

$$\sigma^2_z \sim \left(\sum_{i=1}^{d_1} \underbrace{\sigma^2_x \cdot l^2 \cdot \chi^2(1, 0^2)}_{\text{II}} + \underbrace{\sigma^2_w \cdot \sigma^2_{\mu_x} \cdot \chi^2(1, \mu^2_{\mu_x})}_{\text{III}} \right) + \underbrace{d_1 \cdot \sigma^2_w \cdot \sigma^2_x}_{\text{I}}$$

and its expected value is

$$\begin{aligned}
\mathbb{E}[\sigma^2_z] &= \left(\sum_{i=1}^{d_1} \sigma^2_x \cdot l^2 \cdot (1 + 0^2) + \sigma^2_w \cdot \sigma^2_{\mu_x} \cdot (1 + \mu^2_{\mu_x}) \right) + d_1 \cdot \sigma^2_w \cdot \sigma^2_x \\
&= d_1 \cdot \left(\sigma^2_x \cdot l^2 + \sigma^2_w \cdot \sigma^2_{\mu_x} \cdot (1 + \mu^2_{\mu_x}) + \sigma^2_w \cdot \sigma^2_x \right) \\
&= \underbrace{d_1 \cdot \sigma^2_x \cdot l^2}_{\text{II}} + \underbrace{d_1 \cdot (\sigma^2_{\mu_x} \cdot (1 + \mu^2_{\mu_x}) + \sigma^2_x) \cdot \sigma^2_w}_{\text{I+III}}.
\end{aligned}$$

As σ^2_w has to be positive, we conclude that if we choose $\mathbb{E}[\sigma^2_z] > d_1 \cdot \sigma^2_x \cdot l^2$, then we can set

$$\sigma^2_w = \frac{\mathbb{E}[\sigma^2_z] - d_1 \cdot \sigma^2_x \cdot l^2}{d_1 \cdot (\sigma^2_{\mu_x} \cdot (1 + \mu^2_{\mu_x}) + \sigma^2_x)}.$$

We know (or choose) d_1, l^2, and $\mathbb{E}[\sigma^2_z]$, but we require values for σ^2_x, $\mu^2_{\mu_x}$, and $\sigma^2_{\mu_x}$ to be able to choose σ^2_w. We will find empirical values for these parameters in the next section.

Empirical Parameters + LeakyReLU

To inform the choice of the prior variances of the inner linear layers, we also need to analyze LeakyReLU. We assume the network is an MLP that alternates between linear layers and LeakyReLU. As the message equations of LeakyReLU are too complicated for analysis, we instead use empirical approximation. Let $m_a = \mathcal{N}(\mu_a, \sigma^2_a)$ be an incoming message (from the pre-activation variable to LeakyReLU). We assume that $\sigma^2_a = t$ is a constant and that $\mu_a \sim \mathcal{N}(0, 1)$ is a random variable. By sampling multiple means and then computing the outgoing messages (after applying LeakyReLU), we can approximate the average variance of the outgoing messages, as well as the average and empirical variance over means of the outgoing messages.

We computed these statistics for 101 different leak settings with 100 million samples each, and found that the relationship between leak and μ_{μ_x} (average mean of the outgoing message) is approximately linear, while the relationships between leak and $\sigma^2_{\mu_x}$ or $\mu_{\sigma^2_x}$ are approximately quadratic. Using these samples, we fitted coefficients with an error margin below $5 \cdot 10^{-5}$. For our network, we chose a target variance of 1.5 and a leak of 0.1, resulting in

$$\sigma^2_x = 0.8040586726631379$$
$$\sigma^2_{\mu_x} \cdot (1 + \mu^2_{\mu_x}) = 0.44958619556324186.$$

These values are sufficient for now setting the prior variances of the inner linear layer according to the equations above. Finally, we set the prior variance of the biases to 0.5, so that the output of each linear layer achieves an overall target prior predictive variance of approximately $t = 1.5 + 0.5 = 2.0$.

Results in Practice

In practice, we found that the variance of the predictive posterior still goes up exponentially with the depth of the network despite our derived prior choices. However, if we lower the prior variance further to avoid this explosion, the network is overly restricted and unable to obtain a good fit during training. We therefore set the prior variances as outlined here, but acknowledge that choosing a good prior is a general known problem.

E Evaluation: Analysis on Tabular Data

E.1 Goal, Setup & Methods

In thefollowing, we benchmark the proposed Bayesian neural network (BNN) approach using approximate message passing on a suite of classical regression tasks yet. It is crucial to understand the behavior of BNNs on classic regression tasks of medium difficulty. Our goal is to assess the strengths and weaknesses of BNNs, especially regarding performance and overfitting behavior.

We will analyze if our Bayesian neural networks keep their promise of delivering a good calibration, in particular if the estimated uncertainty matches the errors we are seeing. We are using a suite of regression problems mainly from the widespread UCI machine learning repository Dua and Graff [2017]. This repository contains hundreds of publicly available datasets that are used by researchers as standard benchmarks to test new algorithm approaches. We focus on regression tasks with up to 20 features. Here are the datasets used in our analysis:

- California Housing: The California Housing dataset comprises 8 numerical features derived from the 1990 U.S. Census data. The aim is to estimate the median house value in a specific area, based on nine features with information about the neighborhood. These features include median income, median house age, total number of rooms, total number of bedrooms, population, number of households, latitude, longitude, and a categorical variable for ocean proximity. It has the options “near bay” (San Francisco Bay), “near ocean”, “less than one hour to the ocean”, and “inland”. Challenges in modeling this dataset involve capturing non-linear relationships and spatial dependencies, as well as influencing factors that are not included in the feature set. Nugent [2017]
- Abalone: The Abalone dataset is about predicting the age of these specific snakes by measurements. It contains 4,177 instances with 8 input features: one categorical feature (sex) and seven continuous features (length, diameter, height, whole weight, shucked weight, viscera weight, and shell weight). The target variable is the number of rings, which correlates with the age of the abalone. A significant challenge is the non-linear relationship between the physical measurements and age, as well as the presence of outliers and multicollinearity among features. Nash et al. [1994]
- Wine Quality: This dataset includes two subsets related to red and white "Vinho Verde" wines from Portugal, each with 11 physicochemical input variables such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol. All of these variables are continuous. Furthermore, there is one binary variable indicating if the sample is a white or red whine. The target variable is the wine quality score (0–10) rated by wine tasters. Challenges include class imbalance, as most wines have medium quality scores, and the subjective nature of the quality ratings. Cortez et al. [2009]
- Bike Sharing: The Bike Sharing dataset is about predicting the usage of rental bikes in an area based on seasonal information, weather, and usage profiles. Specifically, it contains contains hourly and daily counts of rental bikes in the Capital Bikeshare system from 2011 to 2012, along with 12 features including season, year, month, day, weekday, hour, holiday, working day, weather situation, temperature, "feels like" temperature, humidity, wind speed, number of casual users, and number of registered users. The target variable is the count of total rental bikes. Modeling challenges involve capturing complex temporal patterns, handling missing data, and accounting for external factors like weather and holidays. Fanaee-T [2013]
- Forest Fires: This dataset comprises 517 instances with 12 features: spatial coordinates (X, Y), temporal variables (month, day), and meteorological data (FFMC, DMC, DC, ISI, temperature, relative humidity, wind, and rain). The target is to predict the burned area of the forest (in hectares) in the northeast region of Portugal in wild fires. The primary challenge is the high skewness of the target variable, with many instances having a burned area of zero, making it difficult to model and evaluate performance accurately. Cortez and Morais [2007]
- Heart Failure: The Heart Failure Clinical Records dataset includes 299 patient records with 13 clinical features such as age, anaemia, high blood pressure, creatinine phosphokinase, diabetes, ejection fraction, platelets, serum creatinine, serum sodium, sex, smoking, time,

and death event. The target variable is a binary indicator of death occurrence. Challenges include the small dataset size, potential class imbalance, and missing values, which can affect the generalizability of predictive models. hea [2020]

- Real Estate Taiwan: This dataset is about predicting house prices in New Taipei City, Taiwan. It contains 414 instances with 6 features: transaction date, house age, distance to the nearest MRT station, number of convenience stores, latitude, and longitude. The target variable is the house price per unit area. Challenges in modeling this dataset involve capturing the influence of location-based features and dealing with a low sample size. Yeh [2018]

Table 3: Summary of Regression Datasets

Dataset	**Samples**	**Features**	**Feature Types**
California Housing	20,640	9	Numerical: latitude, longitude, house median age, total rooms, total bedrooms, population, households, median income; categorical: ocean proximity
Abalone	4,177	8	1 categorical (sex), 7 numerical: length, diameter, height, whole weight, shucked weight, viscera weight, shell weight.
Wine Quality (Red)	1,599	11	All numerical: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol.
Wine Quality (White)	4,898	11	Same as red wine dataset.
Bike Sharing	17,379	12	Mix of categorical and numerical: season, year, month, hour, holiday, weekday, working day, weather situation, temperature, feels-like temperature, humidity, wind speed.
Forest Fires	517	12	2 categorical (month, day), 10 numerical: FFMC, DMC, DC, ISI, temperature, relative humidity, wind, rain, X, Y coordinates.
Heart Failure	299	13	Mix of binary and numerical: age, anaemia, high blood pressure, creatinine phosphokinase, diabetes, ejection fraction, platelets, serum creatinine, serum sodium, sex, smoking, time, death event.
Real Estate Taiwan	414	6	All numerical: transaction date, house age, distance to nearest MRT station, number of convenience stores, latitude, longitude.

For a quick comparison, the number of features and samples are shown in Table 3. As we can see, the datasets have very different sizes. This is an important challenge where we want to evaluate the Bayesian neural network.

For our training, we normalize the values in all columns. In particular, for each column c, we subtract the mean of c from each entry, and divide by the empiric standard deviation. This applies to both feature and target columns. The primary reason for us to do this is the fact that the BNN is designed to handle input with a mean of zero and a standard deviation of one best. Standard neural networks perform best with numbers in this range as well. Furthermore, normalization helps with the common problem that different columns have values in different orders of magnitude before normalization, for example tens of thousands for yearly income, and small numbers for number of bathrooms in the case of California housing. An additional advantage is that performance can be compared across datasets (approximately), highlighting strengths and weaknesses across different settings.

On these datasets, we apply a random split into training and test dataset, where we dedicate 80% on the training dataset and 20% on the test dataset. Then, we run a pipeline where we evaluate the performance on for each dataset both on the Bayesian neural network implemented in Julia, and on the standard neural network implemented in PyTorch.

The Julia implementation uses a neural network with two hidden layers, and 64 neurons per layer. LeakyReLy is the activation function, and the default standard deviation at the last layer is set to 0.4.

The PyTorch network uses two hidden layers with 64 neurons per hidden layer as well. The only difference in terms of architecture to the BNN is the fact that stardard ReLu is used. We train with the same learning rate of $3 \cdot 10^{-3}$. When training with these static datasets which are relatively small, overfitting is a huge challenge. It is especially the challenge of overfitting, that should be addressed by our Bayesian neural networks. As the results will show, overfitting is a significant problem on these datasets for the PyTorch network. To tackle overfitting, several regularization techniques have been proposed. However, overfitting remains a fundamental flaw of the classical neural networks. To aim for a fair comparison, we add a weight-decay regularization to the default setup of the PyTorch networks. The specific weight decay parameter is $1 \cdot 10^{-4}$.

For both candidates, we use a batch size of 256. Training is done over a horizon of 500 epochs which corresponds to very different training lengths, due to the different sizes of the datasets. Hyperparameter tuning was done for none of the datasets.

The primary metric to rate performance on our regression datasets is the root mean squared error between the labels and the predicted values. Because we have a standardized output, trivial benchmarks like the constant zero function give an RMSE of one. Hence, we expect RMSE values from the model to substantially improve over one.

E.2 Results

When running the training an evaluation pipeline, we measure the RMSE of the train and validation dataset after each batch. That naturally comes with small fluctuations, and a slightly uneven-looking learning curve. On the larger datasets, that implies a substantially lower variance of the loss estimation for the validation dataset, and a much smoother learning curve. On the smaller datasets, the evaluation set is quite small, and therefore, the variance is high.

Dataset	BNN Train	BNN Val	PyTorch Train	PyTorch Val	BNN Val / PyTorch Val (%)
Abalone	0.5748	0.5965	0.4643	0.6668	89.46%
Wine Quality	0.7682	0.8095	0.3198	0.7722	104.83%
California Housing	0.5455	0.5764	0.3427	0.4278	134.73%
Bike Sharing	0.2219	0.2432	0.1557	0.216	112.58%
Forest Fires	1.0537	1.2015	0.0302	0.4113	292.14%
Heart Failure	0.9957	0.9605	0.0034	0.9107	105.47%
Real Estate Taiwan	0.6229	0.5823	0.1438	0.5431	107.22%

Table 4: Comparison of minimum RMSE for BNN (Julia) and PyTorch approaches. The data was obtained by running the respective training scripts for 500 epochs and measuring the root mean squared error on training and validation splits.

In Table 4, the best performances on the datasets along a training run are reported. For each of the metrics, we are taking the minimum value over all trained batches. Due to the variance of these estimations, the minimum underestimates the true minimal loss. But the calculation and batch sizes are the the same for both setups, training and loss, so it does not affect the viability of the comparison. Note that the minimum for train and test are not necessarily obtained at the same step.

As the comparison column in Table 4 show, the minimal root mean squared errors are similar for many datasets. On *wine quality*, and *California housing*, *bike sharing*, and *real estate Taiwan*, the PyTorch neural network has advantages in terms of pure regression performance, but only for California

housing this difference is considered strong. On the other datasets, PyTorch's performance advantage is only slight, and within the range of hyperparameter tuning. As we will see in the later analysis, the BNN has not finished to learn after 500 epochs. Our experiment for 5,000 epochs reveils that the performance difference actually shrinks to *x* percent. On Abalone, the BNN's pure performance is even superior. For *forest fires*, we see that the Bayesian neural network was not able to solve the regression problem in this setup (RMSE is larger than 1), but PyTorch achieved respectable predictive power. Neither of the two models was capable to actually solve the *heart failure* dataset, where both models gave only slightly better RMSE performance than one. The detailed learning curves for these four datasets with similar performance are plotted in Figure 5.

A general trend that is visible both in Table 4 and in Figure 5 is the fact BNNs show a minimal level of overfitting, even without further regularization. Depending on the dataset, PyTorch shows high or very high levels of overfitting, even though regularization was applied.

As one would expect, PyTorch's overfitting problem is the least serious on California housing and bike sharing (Figure 5a and 5d), the datasets with the most available samples (California housing has 16,512 training samples, and bike sharing has 13,903 training samples). Over the course of the training, the root mean squared error of the validation and train dataset decrease together, but at around 1/5 of the entire training run, the validation loss stalls, and the training loss decreases further – the model is just overfitting.

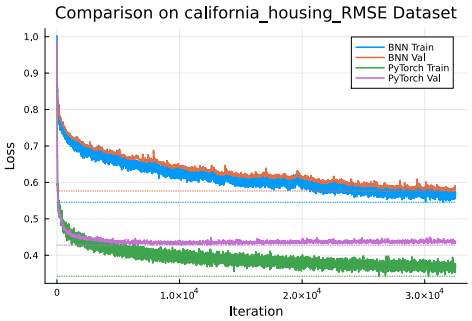
The BNN's training on these datasets stays above PyTorch's learning curves after the rapid first initial improvement, and slowly improves towards PyTorch's level. While it matches PyTorch's performance on bike sharing after 500 epochs, it still has a performance disadvantage on California Housing of 34%. Nevertheless, with additional training, this performance difference shrinks down to only x percent after 5,000 epochs, a very respectable performance. However, the most notable feature on these two training runs is the observation that the BNN only shows minimal overfitting. The training and validation curve decrease on-par, the existing and expected slight overfitting does not increase by time. The minimal train RMSE is 0.546 compared to a validation RMSE of 0.576 on California housing in the first 500 epochs, only 5%! On PyTorch, we see 20% overfitting with increasing trend. On bike sharing, the BNN overfits by 9% compared to 28% for the PyTorch network. The fact that overfitting is not more severe on these datasets is thanks to the large number of samples, around three to four times more than the number of parameters of the model.[7]

Wine quality and abalone both have roughly the same sample size as number of parameters. Therefore, we would expect that overfitting becomes a much more severe issue for these datasets. As Figure 5b and 5c show, the overfitting problem on these two datasets has increased to a severe level. In both cases, the overfitting starts right from the beginning, with a steady improvement of the train loss, but a validation loss that soon finds its minimum, and increases again. In the case of wine quality, this behavior is significant, but stable and smooth. On abalone however, the validation loss behaves very unstable, and never comes close to the BNN's performance.

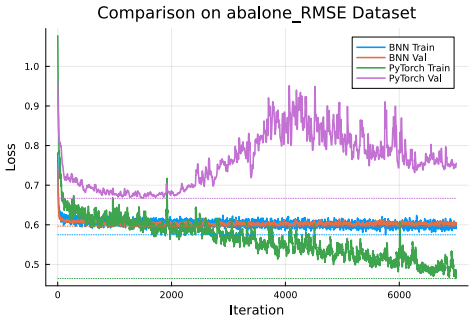
The BNN gets close to its optimal performance very rapidly in both cases. The overall training is much faster than for California housing. Again, we see minimal levels of overfitting, a behavior that is very similar to the analysis of the previous two datasets. Moreover, the existing overfitting does not increase with time but stays at a similar level throughout the training horizon of 500 epochs.

The last three datasets, forest fires, heart failure and real estate Taiwan, are quite important, and also difficult to analyze. They only have a few hundred data points, and tend to be very imbalanced. On the UCI repo and in the literature, they are marked as very difficult datasets. For example, most samples in forest fires contain a fire with zero acres burned, and only very few samples with a high amount of burned wood. Additionally, the validation datasets get so small, that biases could easily find their way into the validation dataset. The concrete performance numbers must therefore be treated with care, and the strange effect of smaller validation loss than train loss can be observed for both PyTorch and BNN on some of the runs. For example, it seems very likely that PyTorch's good performance on forest fires was obtained randomly by lucky parameter initialization instead of actual performance. Hence, based on Figure 5e, Figure 5f, and Table 4, we consider the forest fire dataset and the heart failure datasets unsolved by both models. As we would expect for a model with more than 10x more parameter than training samples, the train loss goes to zero for the PyTorch model.

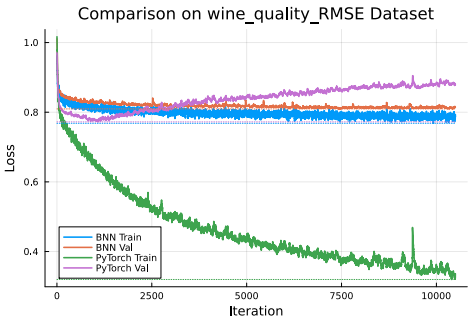
[7] The models have two fully connected hidden layers, one transition to the first layer, and the transition to the regression output. Hence, the models have $64 + 64^2 + 64 = 4224$ weights.



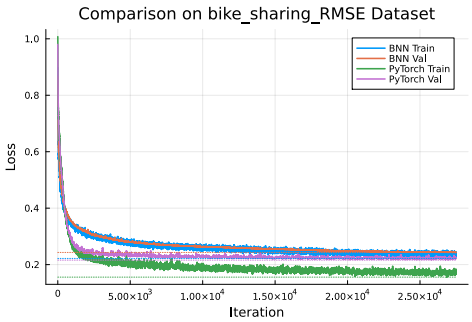
(a) California Housing



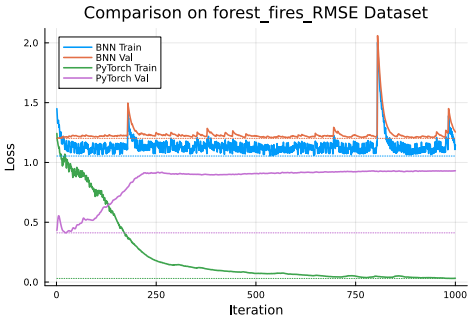
(b) Abalone



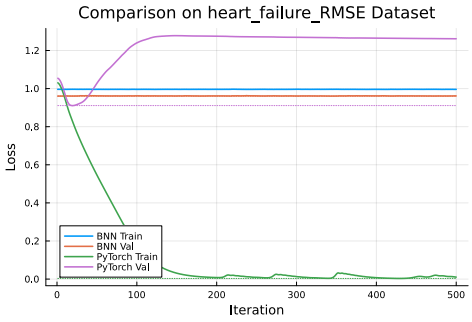
(c) Wine Quality



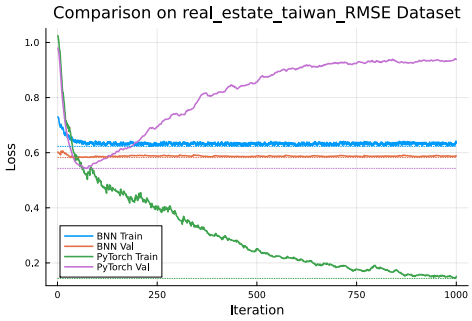
(d) Bike Sharing



(e) Forest Fires



(f) Heart Failure



(g) Real Estate Taiwan

Figure 5: Learning curves of approximate message-passing Bayesian neural networks against PyTorch neural networks

The Bayesian neural network is unable to learn the datasets as well, but it does not overfit. Its training loss never creates the impression that the loss was any lower than it actually is.

The dataset on estimating Taiwanese real estate (learning curve Figure 5g) is the only of the small datasets that actually gets solved to an acceptable level by both models. Again, we see very heavy overfitting by the PyTorch model, and no overfitting by the BNN. (Actually, this is one of the cases where the validation loss is lower than the train loss, probably due to a biased train/test split.) Moreover, the BNN achieves this performance after only a few epochs.

From this analysis, we can note the following learnings:

1. Our approximate message-passing Bayesian neural networks can achieve similar performance like PyTorch neural networks with the same architecture. In most cases, they stay slightly behind in terms of raw performance, but sometimes outperform the standard implementation.
2. Our Bayesian neural networks apparently do not share the fundamental flaw of overfitting. Their train loss often is slightly lower than the validation loss, as it is expected from any type of machine learning model, but the train and validation loss do not detach and differences in these two curves remain low.
3. Bayesian neural networks learn fast on small datasets, and learn slower on larger datasets like California housing.

The last point is worth some deeper investigation. Why does the model learn the Taiwanese real estate after a few epochs, and requires many more epochs for California housing, although one single epoch is already 40 times larger than on the Taiwanese real estate dataset? Bayesian model's learning gets significantly slower over time. While the learning rate remains constant on the standard PyTorch implementation, the speed of change of the weights in the BNN decreases with the variances of the weights. And the variances decrease once more data has been learned. Therefore, a sample at the end of a large dataset has less power to change the model's parameters, and learning slows down. As a conclusion, the BNNs are especially valuable when only few training samples are available, and when overfitting should be avoided.

E.3 Can the BNNs estimate their own uncertainty?

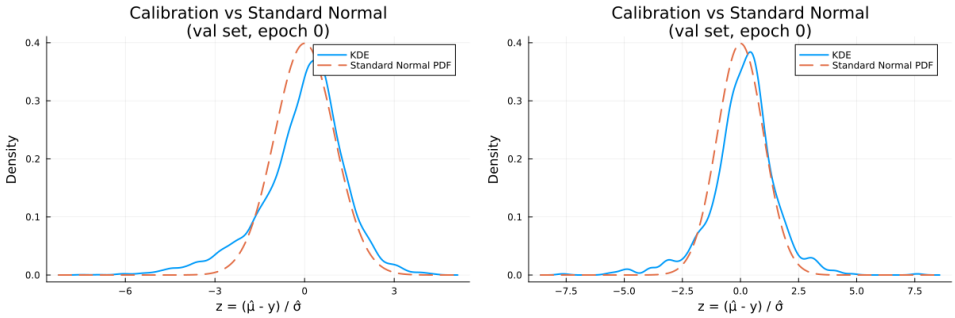
The absence of overfitting on BNNs is a side product of the metric that Bayesian methods traditionally try to optimize, the calibration. In contrast to classic neural networks, our BNNs output their mean $\mu(x)$ together with an estimated standard deviation $\sigma(x)$ expressing the uncertainty for an input x. We can use the ground truth y_x to analyze how well the uncertainty was estimated. Because the ground-truth values were normalized during pre-processing, we expect the μs to also have a mean close to zero, with a variance of roughly one.[8] Specifically, we can calculate the z-score of the observation as

$$z(x) = \frac{\mu(x) - y_x}{\sigma(x)}. \tag{7}$$

If the model was perfectly calibrated, these z-scores would follow a perfect standard normal distribution. That does not mean that the model perfectly predicts the ground truth, and it also does not mean that the uncertainty estimation is correct every single time. Instead, it means that the errors follow the same distribution as predicted by the model. As a rule of thumb, in 68% of the cases, y_x should be within $[\mu(x) - \sigma(x), \mu(x) + \sigma(x)]$, in 95% of the cases within $[\mu(x) - 2\sigma(x), \mu(x) + 2\sigma(x)]$, and in 99% within $[\mu(x) - 3\sigma(x), \mu(x) + 3\sigma(x)]$.

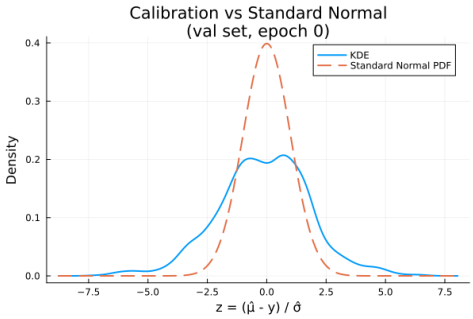
In our experiments, we take the model after 500 epochs for each of the datasets, and calculate the z-scores. We can blindly take the last model because we do not see decreasing performance or problematic overfitting for the BNNs. Then, we use kernel density estimation with Gaussian kernels to obtain an empirical error distribution that is compared to the standard normal distribution. To systematically compare the two distributions, we could report the approximate KL divergence. However, few of our readers have an intuitive understanding of what specific KL numbers mean, and we do not have a comparison partner. Hence, we illustrate the distributions in Figure 6 to best communicate the calibration of the BNN.

[8] Of course, the empiric variance of these μs should not be mixed up with $\sigma(x)$, which expresses a completely different concept, and it is usually much smaller than zero.

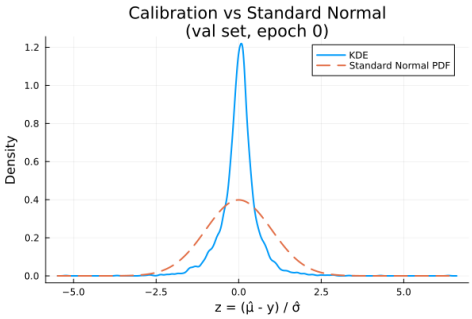


(a) California Housing

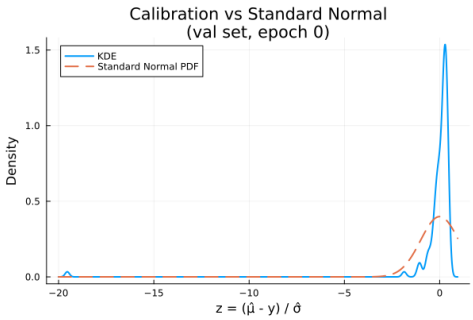
(b) Abalone



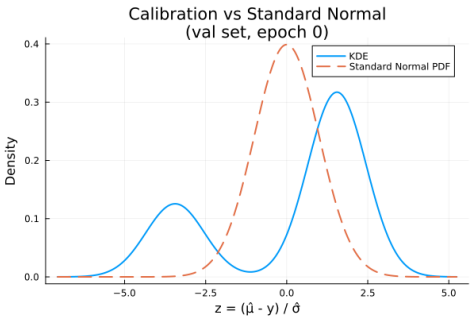
(c) Wine Quality



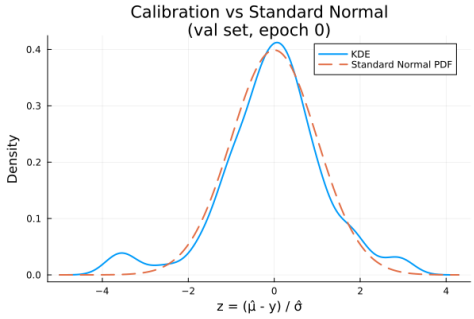
(d) Bike Sharing



(e) Forest Fires



(f) Heart Failure



(g) Real Estate Taiwan

Figure 6: Calibration of the BNN: For each of the datasets, we plot the empiric z-score distribution and a standard normal distribution for reference. All plots are obtained for the validation dataset.

As we can see, the calibration is quite good on most of these datasets. When it comes to California housing (Figure 6a), and abalone (Figure 6b), the uncertainty of the model matches the errors quite well. On wine quality (Figure 6c), the model underestimates the errors, but the uncertainty estimation is still usable. The opposite is true for the bike sharing dataset. As previously discussed, the model solves this dataset very well, and the predicted uncertainties underestimate the true errors (Figure 6d). The wine quality dataset and the bike sharing dataset cannot be solved by the BNN. Hence, the error distributions are rather weak (Figure 6e and 6d). In contrast, we see very strong calibration on the Taiwanese real estate dataset (Figure 6g), although there are a few more outliers than included in the distribution.

We can conclude that the BNN manages the task of uncertainty prediction quite well. Hence, we recommend it to practitioners who are not satisfied with just a prediction, but also want a well-calibrated uncertainty estimation.

F Tables of Message Equations

In the following, we provide tables summarizing all message equations used throughout our model. The tables are divided into three categories: linear algebra operations (Table 5), training signals (Table 6), and activation functions (Table 7). Each table contains the relevant forward and backward message equations, along with illustrations of the corresponding factor graph where necessary. These summaries serve as a reference for the mathematical operations performed during inference and training, and they will be valuable for factor graph modeling across various domains beyond NNs.

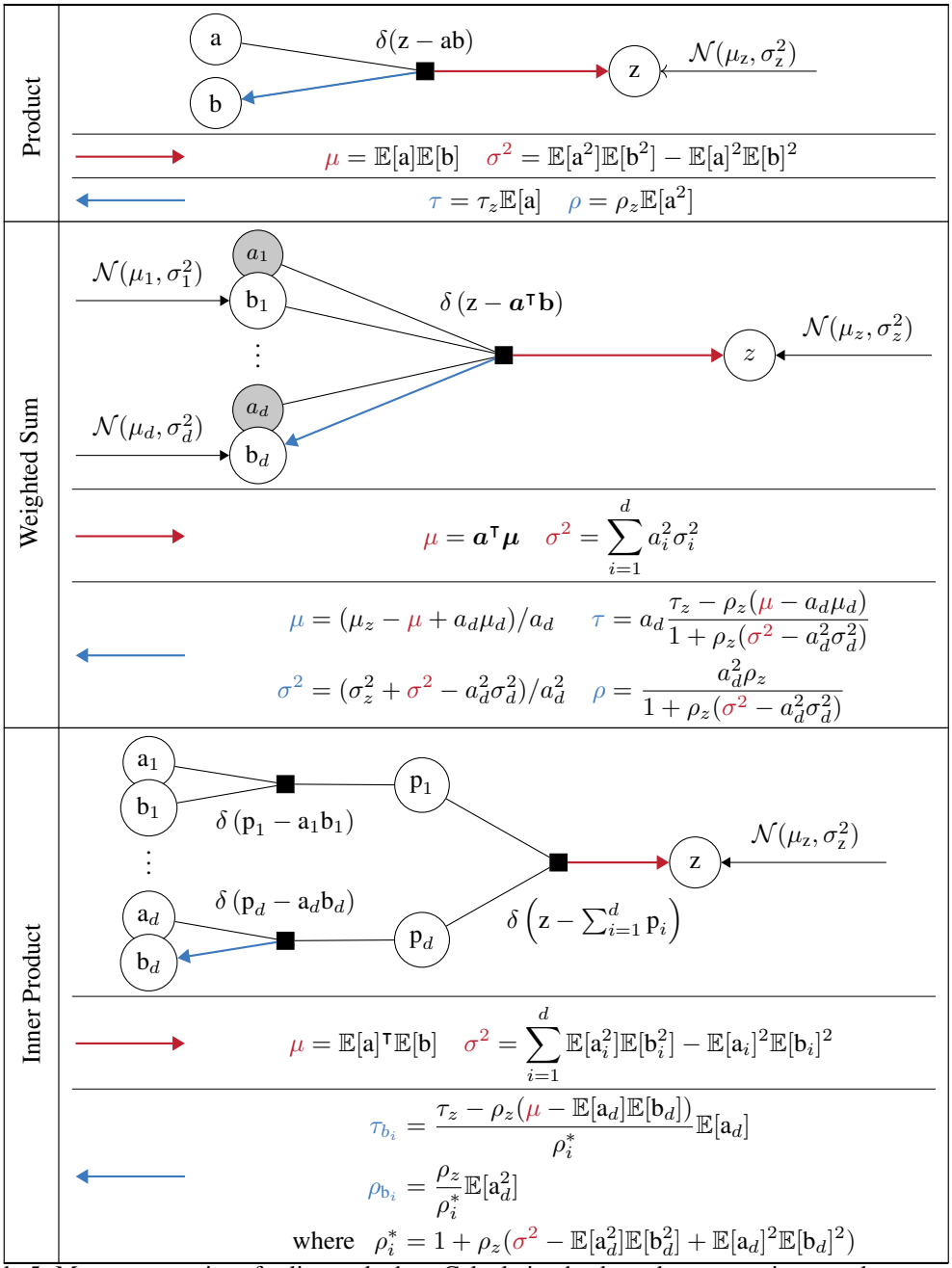
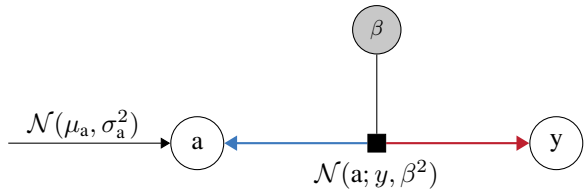
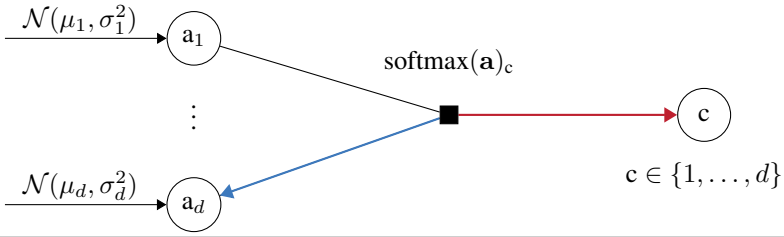


Table 5: Message equations for linear algebra: Calculating backward messages in natural parameters is preferable as it handles edge cases like $a_d = 0$ or $\rho_z = 0$ where location-scale equations are ill-defined. This approach also enhances numerical stability by avoiding division by very small quantities. Note that the inner product messages are simply compositions of the product and weighted sum messages with $a_i = 1,\ i = 1, \ldots, d$.

<table>
<tr><td>Regression</td><td>


$\mu = \mu_\mathrm{a} \quad \sigma^2 = \sigma_\mathrm{a}^2 + \beta^2$

$m_{f\to \mathrm{a}}(\mathrm{a}) = \begin{cases} \mathcal{N}(\mathrm{a}; y, \beta^2) & \text{if y is known (training branch)} \\ 1 & \text{if y is unknown (prediction branch)} \end{cases}$
</td></tr>
<tr><td>Softmax</td><td>


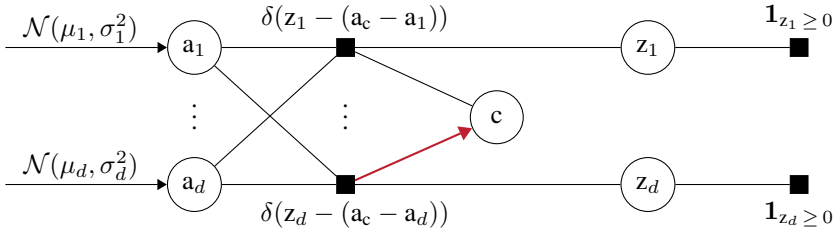
$m_{f\to \mathrm{c}}(i) = \int \mathrm{softmax}(\mathbf{a})_i \mathcal{N}(\mathbf{a}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma})^2)\, d\mathbf{a}$

$\approx \mathrm{softmax}(\boldsymbol{t})_i$ (Daxberger et al. [2022])

where $t_j = \dfrac{\mu_j}{1 + \frac{\pi}{8}\sigma_j^2}$ and $j = 1, \ldots, d$

$m_{f\to \mathrm{a}_d}(\mathrm{a}_d) = \dfrac{\mathcal{N}(\mathrm{a}_d; m_1/m_0, m_2/m_0 - (m_1/m_0)^2)}{m_{\mathrm{a}_d \to f}(\mathrm{a}_d)}$

where $m_k = \int_{\mathrm{a}_d} \mathrm{a}_d^k \mathcal{N}(\mathrm{a}_d; \mu_d, \sigma_d^2)\, \mathrm{softmax}(t_1, ..., t_{d-1}, \mathrm{a}_d)_\mathrm{c}\, d\mathrm{a}_d$

is approximated via numerical integration
</td></tr>
<tr><td>Argmax</td><td>


$\int \mathbf{1}_{\mathrm{z}_d \geq 0} \delta(\mathrm{z}_d - (\mathrm{a}_\mathrm{c} - \mathrm{a}_d)) \mathcal{N}(\mathrm{a}_\mathrm{c}; \mu_\mathrm{c}, \sigma_\mathrm{c}^2) \mathcal{N}(\mathrm{a}_d; \mu_d, \sigma_d^2)\, d\mathrm{a}_\mathrm{c} d\mathrm{a}_d d\mathrm{z}$

$= \begin{cases} 1 & \text{for c} = d \\ \Pr[\mathrm{a}_\mathrm{c} \geq \mathrm{a}_d] = \phi(0; \mu_d - \mu_\mathrm{c}, \sigma_d^2 + \sigma_\mathrm{c}^2) & \text{for c} \neq d \end{cases}$

If c is known, many edges become constant and can be omitted. Assume w.l.o.g. $\mathrm{c} = d$, then a_d is connected to $d - 1$ factors and all other a_i to only one each. The messages to $\mathrm{a}_1, \ldots, \mathrm{a}_d$ follow from the weighted sum factor, given Gaussian approximations of the messages from z_i. We derive these by moment-matching the marginals of z_i (see Building Block 2) and dividing by the message from the weighted sum factor. To stabilize training, we regularize the variance of $m_{f\to \mathrm{a}_i}$ by a factor of $\phi(0; \mu_\mathrm{c} - \mu_i, \sigma_i^2 + \sigma_\mathrm{c}^2)$ and multiply $m_{f\to \mathrm{a}_i}(\mathrm{a}_i)$ by $\mathcal{N}(\mathrm{a}_\mathrm{c}; 1 \text{ if } i = \mathrm{c} \text{ else } -1, \gamma^2)$, effectively mixing in one-hot regression factors during training.
</td></tr>
</table>

Table 6: Message equations for training signals. Note that the backward messages only apply in the case in which the target is known, i.e., on the training branches. On the prediction branch we only do foward passes.

<table>
<tr><td>Auxiliary Equations</td><td>

$$\text{ReLUMoment}_k(\mu, \sigma^2) = \begin{cases} \mathbb{E}[\text{ReLU}(\text{a})] \text{ with a} \sim \mathcal{N}(\mu, \sigma^2) & \text{for } k = 1 \\ \mathbb{E}[\text{ReLU}^2(\text{a})] \text{ with a} \sim \mathcal{N}(\mu, \sigma^2) & \text{for } k = 2 \end{cases}$$

$$= \begin{cases} \sigma\varphi(x) + \mu\phi(x) & \text{for } k = 1 \\ \sigma\mu\varphi(x) + (\sigma^2 + \mu^2)\phi(x) & \text{for } k = 2 \end{cases}$$

where φ and ϕ denote the pdf and cdf of $\mathcal{N}(0, 1)$, respectively.

$$\zeta_k(\mu_1, \sigma_1, \mu_2, \sigma_2) := \int_0^\infty \text{a}^k \mathcal{N}(\text{a}; \mu_1, \sigma_1^2)\mathcal{N}(\text{a}; \mu_2, \sigma_2^2)\, d\text{a}$$

$$= \mathcal{N}(\mu_1; \mu_2, \sigma_1^2 + \sigma_2^2) \cdot \begin{cases} \text{ReLUMoment}_k(\mu_m, \sigma_m^2) & \text{for } k = 1, 2 \\ \phi(\mu_m/\sigma_m) & \text{for } k = 0 \end{cases}$$

$$\text{with } \tau_m = \frac{\mu_1}{\sigma_1^2} + \frac{\mu_2}{\sigma_2^2}, \quad \rho_m = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}, \quad \mu_m = \frac{\tau_m}{\rho_m}, \quad \text{and } \sigma^2 = \frac{1}{\rho_m}$$

See Building Block 1 for the derivation of this equation.

</td></tr>
<tr><td>LeakyReLU</td><td>

We use marginal approximation while:

1. The outputs are finite and not NaN
2. Forward message: Precision of $m_{f\to\text{z}}$ is $\geq$ precision of $m_{\text{a}\to f}$, and $m_0 > 10^{-8}$
3. Backward message: It has worked well to require $(\tau_\text{z} > 0) \vee (\rho_\text{z} > 2 \cdot 10^{-8})$

Otherwise, we fall back to direct message approximation (forward) or $\mathbb{G}(0, 0)$ (backward).

</td></tr>
<tr><td>Direct</td><td>

$$\mu = (1 - \alpha) \cdot \text{ReLUMoment}_1(\mu_\text{a}, \sigma_\text{a}^2) + \alpha \cdot \mu_\text{a}$$
$$\sigma^2 = (1 - \alpha^2) \cdot \text{ReLUMoment}_2(\mu_\text{a}, \sigma_\text{a}^2) + \alpha^2 \cdot (\sigma_\text{a}^2 + \mu_\text{a}^2) - \mu^2.$$

</td></tr>
<tr><td>Marginal</td><td>

$$m_{f\to\text{z}}(\text{z}) = \frac{\mathcal{N}(\text{z}; \frac{m_1}{m_0}, \frac{m_2}{m_0} - (\frac{m_1}{m_0})^2)}{m_{\text{z}\to f}(\text{z})}$$

$$\text{where} \quad m_k = (-1)^k \cdot \zeta_k(-\mu_a, \sigma_a^2, \ -\alpha \cdot \mu_\text{z}, \alpha^2 \cdot \sigma_\text{z}^2) + \zeta_k(\mu_a, \sigma_a^2, \ \mu_\text{z}, \sigma_\text{z}^2)$$

To compute the marginal backward message, set $\alpha_\text{back} = \alpha^{-1}$
and swap $m_{a\to f}$ and $m_{\text{z}\to f}$ in the equation

</td></tr>
</table>

Table 7: Message equations for LeakyReLU with ReLU as the special case $\alpha = 0$.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We provide a novel method for message passing in Bayesian Neural Networks.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Section 6, page 9.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: See Appendix.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

Justification: We use reproducible problem examples and provide supplemental code.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide supplemental code and will make it open access.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We explain all chosen hyperparameters.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: See Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Given the considered problem, there are no ethical issues.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We point to potential applications of the solution.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: There are no such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [NA]

Justification: We cite relevant work.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: The paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.