
Infinitely Deep Bayesian Neural Networks with Stochastic Differential Equations

Winnie Xu
University of Toronto

Ricky T.Q. Chen
University of Toronto

Xuechen Li
Stanford University

David Duvenaud
University of Toronto

Abstract

We perform scalable approximate inference in continuous-depth Bayesian neural networks. In this model class, uncertainty about separate weights in each layer gives hidden units that follow a stochastic differential equation. We demonstrate gradient-based stochastic variational inference in this infinite-parameter setting, producing arbitrarily-flexible approximate posteriors. We also derive a novel gradient estimator that approaches zero variance as the approximate posterior over weights approaches the true posterior. This approach brings continuous-depth Bayesian neural nets to a competitive comparison against discrete-depth alternatives, while inheriting the memory-efficient training and tunable precision of Neural ODEs.

1 INTRODUCTION

Taking the limit of neural networks to be the composition of infinitely many residual layers provides a way to implicitly define its output as the solution to an ODE (Haber and Ruthotto, 2017; E, 2017). This continuous-depth parameterization decouples the specification of the model from its computation. While the paradigm adds complexity, it has several benefits: (1) Computational cost can be traded for precision in a fine-grained manner by specifying error tolerances for adaptive computation, and (2) memory costs for training can be significantly reduced by running the dynamics backwards in time to reconstruct activations of intermediate states needed for backpropagation.

On the other hand, the Bayesian treatment for neural networks modifies the typical training pipeline where

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

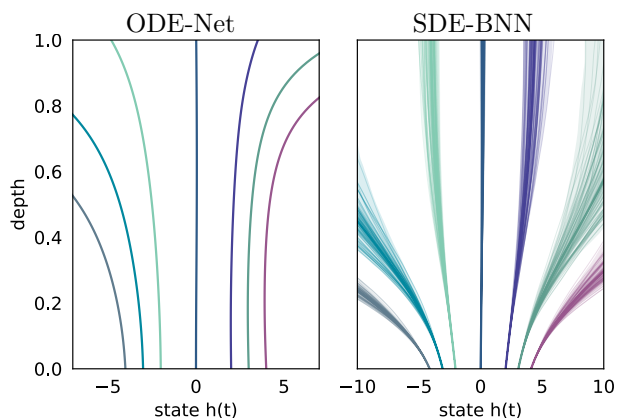


Figure 1: Hidden unit trajectories in an ODE-Net and an SDE-BNN. *Left*: A continuous-depth residual network has deterministic transformations of its hidden units from depths $t = 0$ to $t = 1$. *Right*: Uncertainty in the weights of a Bayesian continuous-depth residual network implies uncertainty in its hidden unit activation trajectories. Shaded regions show densities over samples from the learned posterior dynamics. *Both*: Each distinct color corresponds to a different initial state corresponding to different data inputs.

instead of performing point estimates, a distribution over parameters is inferred. Although this approach adds complexity, it automatically accounts for model uncertainty. In turn, model averaging can be done to combat overfitting and improve calibration, especially on out-of-distribution data (Zhang et al., 2018; Osawa et al., 2019).

How can we combine the benefits of continuous-depth models with those of Bayesian neural networks? The simplest approach is a “Bayesian neural ODE” (Yıldız et al., 2019; Dandekar et al., 2020), which integrates out the finitely-many parameters of a standard neural ODE for prediction.

This approach is straightforward to implement, and can inherit the advantages of both Bayesian and continuous-depth neural nets. However, empirically, standard Gaussian approximate posteriors are a relatively poor

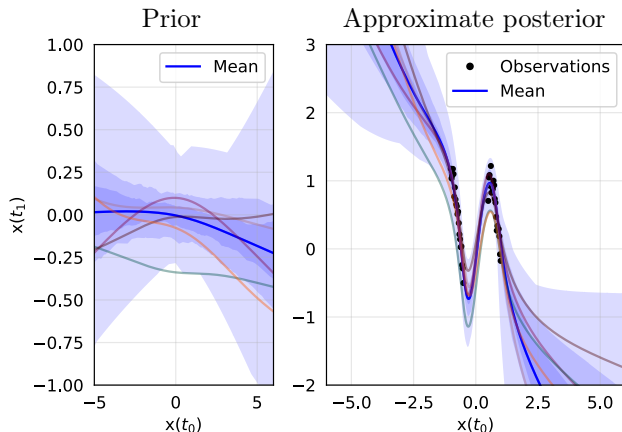


Figure 2: Predictive prior and posterior of the SDE-BNN on a non-monotonic toy dataset. Blue areas indicate density percentiles, and distinct colored lines show model samples.

match for neural ODEs, not to mention the drawbacks of also being used in the prior. Additionally, it does not exploit the special synergy available between continuous-time models and approximate inference.

In this paper, we show that an alternative construction of Bayesian continuous-depth neural networks has additional practical benefits. Specifically, we consider the limit of infinite-depth Bayesian neural networks with separate unknown weights at each layer, a model class that we refer to as SDE-BNNs. We develop a unique network architecture that enhances model expressivity through time-correlated weights and scales linearly, instead of quadratically, with the parameter dimensionality. Combined with our novel formulation of a zero-variance gradient estimator, we show that approximate inference can be realized through the maximization of our modified variational lower bound, effectively scaling up the gradient-based variational inference scheme described by Li et al. (2020) (preliminary forms of which appeared in earlier works (Archambeau et al., 2008; Opper, 2019; Tzen and Raginsky, 2019a)).

With this approach, the state of the output layer is computed by a black-box adaptive SDE solver. Figure 1 contrasts our neural SDE with the neural ODE parameterization. This approach maintains the adaptive computation and constant-memory cost of training Bayesian neural ODEs and adds two unique benefits:

- The variational posterior can be made arbitrarily expressive by simply enlarging the neural network that parameterizes the dynamics of the approximate posterior. Under mild conditions, this approach can approximate the true posterior arbitrarily closely.

- The variational objective admits a variance-reduced gradient estimator that is a natural extension of the “sticking the landing” trick (Roeder et al., 2017). Combined with arbitrarily expressive approximate posteriors, it is consistent and has vanishing variance as the approximate posterior approaches the true.

Notably, our low-variance gradient estimator can also be applied to variational inference in SDEs more generally, such as for time-series modeling, but such applications are beyond the scope of this paper.

2 BACKGROUND

Bayesian Neural Networks Given a dataset, there are often many functions that fit the data well, which a given neural network can express with different parameter values. Instead of making point estimates of the parameters, the Bayesian paradigm frames learning as posterior inference. Predictions are obtained through integrating over many possible parameter settings. Formally, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ and prior distribution over model weights $p(w)$, we want to compute a posterior $p(w|\mathcal{D}) \propto p(\mathcal{D}|w)p(w)$. We can optimize an approximate posterior distribution $q(w)$ that minimizes the Kullback-Leibler (KL) divergence, i.e. maximizing the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{\text{ELBO}}(\phi) = \mathbb{E}_{q(w)} [\log p(\mathcal{D}|w)] - D_{\text{KL}}(q(w)||p(w)). \quad (1)$$

Estimating gradients of this objective using simple Monte Carlo is known as stochastic variational inference (SVI) (Hoffman et al., 2013; Rezende et al., 2014)).

One of the main technical challenges of SVI is choosing a parametric family of approximate posteriors that is tractable to sample from and evaluate, while being flexible enough to approximate the true posterior well. Most scalable inference techniques use Gaussian approximate posteriors with restricted covariance structure between network parameters (Graves, 2011; Blundell et al., 2015; Zhang et al., 2018; Mishkin et al., 2018). Others construct complex approximate posteriors with normalizing flows (Krueger et al., 2018; Louizos and Welling, 2017) or through distillation (Balan et al., 2015; Wang et al., 2018).

Neural Ordinary Differential Equations Neural ordinary differential equations (Chen et al., 2018) define ODEs using neural networks:

$$dh_t = f_\theta(h_t, t) dt, \quad h_0 \in \mathbb{R}^d, \quad (2)$$

where $f : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ is a Lipschitz function defined by a neural network with parameters θ . Starting at an initial value $h_0 = x$ given by a data example and

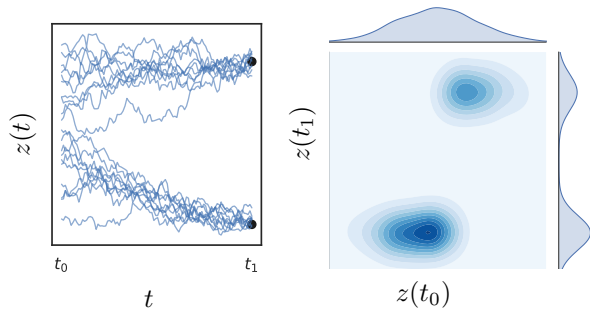


Figure 3: Neural SDEs can learn arbitrarily expressive approximate posteriors. *Left*: Samples from an approximate posterior, trained with an OU prior and conditioned on two observations with Cauchy likelihoods. *Right*: Joint distribution and marginals of the approximate posterior process z at times t_0 and t_1 .

integrating these dynamics forward for a finite time can be seen as passing the input through an infinitely-deep residual network. For learning scalar-valued functions, adding extra dimensions to h and a linear final layer induces similar universal approximations to standard neural networks (Dupont et al., 2019; Zhang et al., 2019b) trained by standard stochastic gradient descent methods. Using adaptive ODE solvers can trade evaluation speed for precision. The adjoint sensitivity method saves memory during training through reconstructing the trajectory of the hidden units h by running the dynamics backwards in backpropagation.

2.1 Latent Stochastic Differential Equations

Informally, an SDE can be viewed as an ODE with infinitesimal noise added throughout time. Formally:

$$dw_t = f_\theta(w_t, t) dt + g_\theta(w_t, t) dB_t, \quad (3)$$

where $w_0 \in \mathbb{R}^d$ is the initial state, $f_\theta : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ and $g_\theta : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^{d \times m}$ are functions Lipschitz in both arguments, dubbed the *drift* and *diffusion*, respectively, and $\{B_t\}$ is an m -dimensional Brownian motion.

Some works have considered training SDEs with dynamics parameterized by neural networks (Li et al., 2020; Tzen and Raginsky, 2019a; Peluchetti and Favaro, 2020b; Innes et al., 2019; Kong et al., 2020; Liu et al., 2019). Note that directly optimizing the drift and diffusion to maximize the average log-likelihood of an observation $\log p(y_t | w_t)$ would result in the diffusion approaching 0, conditional on the ODE fitting the training data well with the diffusion somewhat unconstrained.

Instead of directly optimizing the parameters of an SDE to match the data, a better approach is to use an SDE to define a prior over trajectories of w , and optimize the marginal likelihood of the data, integrating over all

trajectories of w weighted by the prior. Luckily, we can specify an approximate posterior over trajectories using a second SDE. We define the approximate posterior by

$$dw_t = f_\phi(w_t, t) dt + g_\theta(w_t, t) dB_t. \quad (4)$$

When the dynamics of the approximate posterior f_ϕ is parameterized by a neural network, this family of posteriors is extremely expressive. Figure 3 shows that such a variational family can easily approximate non-Gaussian and multi-modal posteriors on path space.

If both the SDE defined by equation 3 and equation 4 share the same diffusion function, then the KL between the two induced measures on path space has the following form (Li et al., 2020; Tzen and Raginsky, 2019a):

$$D_{\text{KL}}(\mu_q || \mu_p) = \mathbb{E}_{q_\phi(w)} \left[\int_0^1 \frac{1}{2} \|u(t, \phi)\|_2^2 dt \right] \text{ where } \quad (5)$$

$$u(t, \phi) = g_\theta(w_t, t)^{-1} [f_\theta(w_t, t) - f_\phi(w_t, t)] \quad (6)$$

where μ_q and μ_p are path space probability measures induced respectively by equation 4 and equation 3, and the expectation is taken under the approximate posterior, denoted $q_\phi(w)$. Intuitively, this KL divergence resembles the summative difference over time horizon $[0, 1]$ between the prior drift f_θ and f_ϕ , scaled by the diffusion. This divergence can be estimated up to a constant with simple Monte Carlo, sampling trajectories from the dynamics given by the approximate posterior.

SDEs as expressive approximate posteriors To ensure that the KL divergence between the prior and approximate posterior on path space is finite, the same diffusion function $g_\theta(w_t, t)$ must be used for the approximate posterior and prior. Surprisingly, this does not limit the expressivity of the approximate posterior. Boué et al. (1998) show that there is a one-to-one correspondence between the space of path measures and drift functions that result in the same path space KL divergence. This implies that any path space measure close to the true posterior can be instantiated by SDEs with appropriate drifts. It follows that an approximate posterior parameterized by a sufficiently expressive family of function approximators can be made arbitrarily close to the true posterior. The Girsanov reparameterization of the variational formula, derived from Boué (Tzen and Raginsky, 2019a, Section 4), proves that the ELBO is tight when the drift is optimal. This means that there exists a ground truth drift function that can make the ELBO tight, the approximation of which can be achieved with a high capacity neural network.

3 INFINITELY DEEP BNNs

Standard discrete-depth residual networks can be defined as a composition of layers of the form:

$$h_{t+\epsilon} = h_t + \epsilon f(h_t, w_t), \quad t = 1 \dots T, \quad (7)$$

where t is the layer index, $h_t \in \mathbb{R}^{D_h}$ denotes a vector of hidden unit activations at layer t , the input $h_0 = x$, and $w_t \in \mathbb{R}^{D_w}$ represents the parameters for layer t . In the discrete setting, $\epsilon = 1, \in \mathbb{R}$.

We can construct a continuous-depth variant of residual networks by setting $\epsilon = 1/T$ and taking the limit as $T \rightarrow \infty$. This yields a differential equation that describes the hidden unit evolution as a function of depth t . Since standard residual networks are parameterized with different layerwise “weights”, we denote them w_t . To specify different weights at each layer with a finite number of parameters, we introduce a hypernetwork f_w that specifies the change in weights as a function of depth and the current weights (Ha et al., 2016). The evolution of the hidden unit activations and weights can then be combined into a single differential equation:

$$\frac{d}{dt} \begin{bmatrix} h_t \\ w_t \end{bmatrix} = \begin{bmatrix} f_h(t, h_t, w_t) \\ f_w(t, w_t) \end{bmatrix} \quad (8)$$

with some learned initial weight value w_{t_0} . Using time-varying weights is similar to augmenting the state (Dupont et al., 2019; Zhang et al., 2019c). See Appendix Figure 8 on the effects of augmentation. We perform Bayesian inference on the weight process w_t , assigning a suitable prior stochastic process and performing variational inference in this infinitesimal limit.

Like all Bayesian neural networks with observation likelihoods, our framework models uncertainty both about parameters and about individual observations: The likelihood $p(y|h_1)$ captures the observational noise, while the SDE encodes weight uncertainty.

Prior process on weights Typical priors for Bayesian neural networks are independent Gaussians across all weights and layers. Taking the infinitesimal limit of such a prior gives a white noise process prior on the weights $w(\cdot)$. However, initializing this noise while maintaining finite variance at scale is difficult (Peluchetti and Favaro, 2020a,b).

Instead, we use the Ornstein–Uhlenbeck (OU) process as the prior on weights. The process is characterized by an SDE with drift and diffusion:

$$f_p(w_t, t) = -w_t, \quad g(w_t, t) = \sigma I_d, \quad (9)$$

respectively, where σ is a hyperparameter. We choose this prior for its simplicity and bounded marginal variance at a constant in the large time limit.

Approximate posterior over weights We parameterize the approximate posterior on weights implicitly using another SDE with the following drift function:

$$f_q(w_t, t, \phi) = \text{NN}_\phi(w_t, t, \phi) - f_p(w_t, t). \quad (10)$$

This drift f_q is parameterized by a small neural network (NN) with parameters ϕ . With this drift, the approximate posterior process will generally have non-Gaussian, non-factorized marginals; its expressive capacity can be increased by making the neural net larger.

Evaluating the network Given an input, we marginalize over weight and hidden unit trajectories. This can be done with simple Monte Carlo, sampling a weight path $\{w_t\}$ from the posterior process and evaluating the network activations $\{h_t\}$ given the sampled weights and input. Both steps require solving a differential equation. Luckily, both can be solved simultaneously with the augmented state SDE:

$$d \begin{bmatrix} w_t \\ h_t \end{bmatrix} = \begin{bmatrix} f_w(w_t, t, \phi) \\ f_h(h_t, t, w_t) \end{bmatrix} dt + \begin{bmatrix} g_w(w_t, t) \\ \mathbf{0} \end{bmatrix} dB_t, \quad (11)$$

where $h_0 = x$, the input. The learnable parameters are the initial weight values at time zero w_0 (either point estimated or inferred) and those of drift function ϕ .

Output likelihood The final state of the hidden units h_1 is used to parameterize the likelihood of the target output y : $\log p(y|x, w) = \log p(y|h_1)$. For instance, $p(y|h_1)$ could be a Cauchy likelihood for regression, or categorical likelihood for classification.

Training objective To fit the network to data, we maximize the lower bound on marginal likelihood given by the infinite-dimensional ELBO:

$$\mathcal{L}_{\text{ELBO}_\infty}(\phi) = \mathbb{E}_{q_\phi(w)} \left[\log p(\mathcal{D}|w) - \int_0^1 \frac{1}{2} \|u(w_t, t, \phi)\|_2^2 dt \right].$$

The sampled weights, the hidden activations, and the training objective are all computed simultaneously with a single call to an adaptive SDE solver. Gradients of the sampled loss can also be efficiently computed using adaptive solvers, following Li et al. (2020).

4 VARIANCE-REDUCED GRADIENTS

Roeder et al. (2017) showed that when optimizing expectations using the reparameterization gradient, a gradient estimator with lower variance can be constructed by removing a score function term that has zero expectation, and that the variance of this gradient estimator approaches zero as the approximate posterior

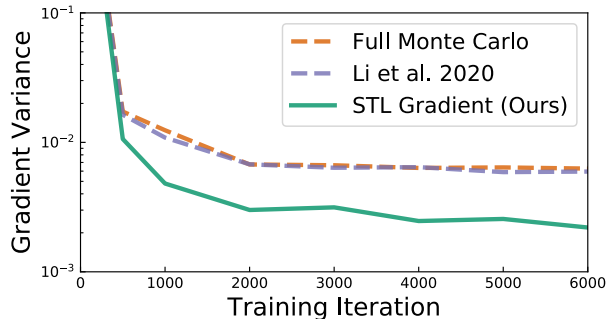


Figure 4: Comparison of the variance in three gradient estimators. On this toy problem, our new gradient estimator reduces variance by a factor of roughly 4.

approaches the true posterior. We refer to this trick as “sticking the landing” (STL). We generalize this to our SDE setting by replacing the original estimator of the path space KL with the following STL estimator:

$$\widehat{\text{KL}}_{\text{STL}} = \int_0^1 \frac{1}{2} \|u(w_t, t, \phi)\|_2^2 dt + \int_0^1 u(w_t, t, \perp(\phi)) dB_t, \quad (12)$$

where $w(\cdot) \sim q_\phi(w)$ and u is defined in equation 6, the path $\{w_t\}_{t \in [0, T]}$ is sampled from the approximate posterior process, and $\perp(\cdot)$ is the stop gradient function that renders the input a constant with respect to which gradient propagation is stopped. Note that 12 is the fully Monte Carlo version referred to in 4 from which our STL variant is derived.

The second term in equation 12 is a martingale and has expectation zero. Therefore, in prior works (Li et al., 2020; Tzen and Raginsky, 2019a,b), Monte Carlo estimation was only performed for the first term, but we find that this approach does not necessarily reduce the variance of the *gradient* (Figure 4).

Because our approximate posterior can be made arbitrarily expressive, we conjecture that our approach can achieve arbitrarily low gradient variance towards the end of training if the f_w parameterization is expressive enough. See Appendix A.2 for a heuristic derivation.

We show the variance of different gradient estimators in Figure 4, averaged across the parameters θ , in a 1D regression setting. We compare STL against a “Full Monte Carlo” estimate which includes the second additional term without gradient stopping, as well as the estimator that was previously used by Li et al. (2020) which ignores the second term. Figure 4 shows that STL obtains lower variance than alternatives, when matching an exponentiated Brownian motion. Table 4 shows training performance improvements.

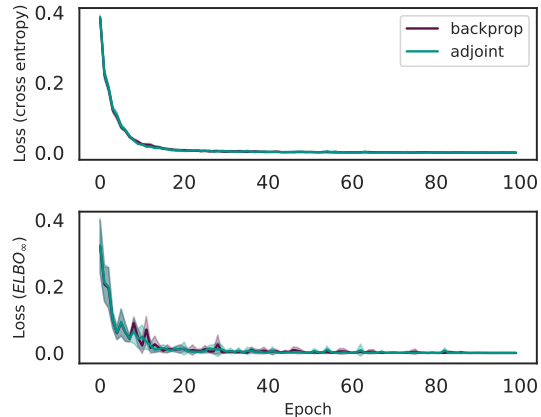


Figure 5: Benchmarking two gradient computation methods: (1) Back-propagation through the SDE solver, and (2) the memory-efficient stochastic adjoint of Li et al. (2020). Both methods have similar optimization dynamics, final performance, and wall-clock time, but the adjoint approach is more memory-efficient. Detailed comparisons of wall-clock time and evaluation step results in Appendix C.8.

5 EXPERIMENTS

We investigate the effectiveness of our proposed approximate inference method for training continuous-depth neural nets, referred to as SDE-BNN, in terms of classification accuracy, calibration, perturbation robustness, and speed-precision trade-offs. Our code is publicly available [here](#) and experimental settings in Table 2.

We consider toy regression and image classification tasks on MNIST and CIFAR-10. We also investigate out-of-distribution generalization. Notably, our approach does not require *post hoc* recalibration methods such as training with temperature scaling (Guo et al., 2017) or isotonic regression (Zadrozny and Elkan, 2002).

Backpropagation through solvers vs. adjoint

We experimented with fixed- and adaptive-step SDE solvers, and the stochastic adjoint of Li et al. (2020). Figure 5 shows similar convergence for both approaches. Appendix C shows that both had similar numbers of dynamics function evaluations and wall-clock time.

The overhead for estimating error in our adaptive solvers was substantial; therefore, for final model evaluation, we trained with fixed-step solvers, where the number of steps is chosen to be large enough to match the convergence speed of our adaptive-step solvers.

Baselines For a fixed-depth network baseline, we compare to standard residual networks. We then test variational inference on the weights of these models.

Table 1: Classification accuracy and expected calibration error (ECE) on MNIST and CIFAR-10. We separate models into point estimates, discrete-time models, and continuous-time models. Our SDE-BNN outperforms other continuous-time Bayesian neural nets (BNNs) and perform competitively against discrete-time BNNs. [†]Results by Izmailov et al. (2021) where a modified residual network architecture was used; only one seed was reported.

Model	MNIST		CIFAR-10	
	Accuracy (%)	ECE ($\times 10^{-2}$)	Accuracy (%)	ECE ($\times 10^{-2}$)
ResNet32	99.46 \pm 0.00	2.88 \pm 0.94	87.35 \pm 0.00	8.47 \pm 0.39
ODEnet	98.90 \pm 0.04	1.11 \pm 0.10	88.30 \pm 0.29	8.71 \pm 0.21
HyperODEnet	99.04 \pm 0.00	1.04 \pm 0.09	87.92 \pm 0.46	15.86 \pm 1.25
MFVI ResNet32	99.44 \pm 0.00	2.76 \pm 1.28	86.97 \pm 0.00	3.04 \pm 0.94
MFVI [†]	—	—	86.48	1.95
Deep Ensemble [†]	—	—	89.22	2.79
HMC (“gold standard”) [†]	98.31	1.79	90.70	5.94
MFVI ODEnet	98.81 \pm 0.00	2.63 \pm 0.31	81.59 \pm 0.01	3.62 \pm 0.40
MFVI HyperODEnet	98.77 \pm 0.01	2.82 \pm 1.34	80.62 \pm 0.00	4.29 \pm 1.10
SDE BNN	99.30 \pm 0.09	0.63 \pm 0.10	89.84 \pm 0.94	7.19 \pm 0.37
SDE BNN (+ STL)	99.10 \pm 0.09	0.78 \pm 0.12	89.10 \pm 0.45	7.97 \pm 0.51

We also perform ablation studies to compare with standard variational inference approaches over continuous-depth networks. Specifically, we compare to a mean field variational inference (MFVI) ODEnet where stochastic variational inference is performed over depth-invariant weights. This baseline is a fully-factorized Gaussian approximate posterior, *i.e.* mean-field approximation, and been used for Neural ODEs by Look and Kandemir (2019); Dandekar et al. (2020).

We further compare our model to a MFVI HyperODEnet, where a learned drift is applied to w , but mean-field inference is instead performed over the parameters of the hypernetwork. Alternatively, one can interpret this as another MFVI ODEnet with a larger state and a more complex drift function but with similar computational complexity to SDE-BNN. This setting contrasts our approach of doing Bayesian inference over the entire continuous-depth network as a stochastic process.

Parameterizing the drift function We parameterized the drift of the variational posterior f_w with a simple multilayer perceptron. To ensure optimization starts at a stable set of dynamics, we subtract the prior drift so that the approximate posterior equals the prior when the final layer is initialized to output zero.

Hyperparameters We swept learning rates in the range [1e-4, 1e-3], selecting the optimal based on the validation set. We train with the default Adam optimizer (Kingma and Ba, 2015). In image classification experiments, all convolutional layers of the drift network are time-conditional and use the `tan`h non-linearity. The diffusion coefficient σ was selected from validation performance over {0.1, 0.2, 0.5}.

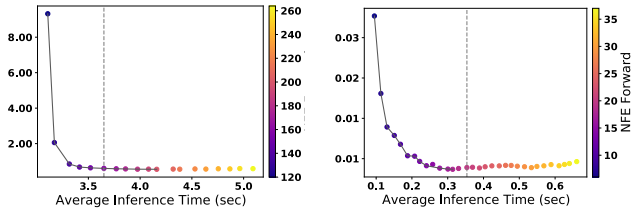
5.1 1D Regression

We first verify the capabilities of the SDE-BNN on a 1D regression problem. Conditioned on a sample from the diffusion process, each sample from a one-dimensional SDE-BNN is a bijective mapping from the inputs to the outputs. This implies that every function sampled from a 1D SDE-BNN is monotonic. To be able to sample non-monotonic functions, we augment the state with 2 extra dimensions initialized to zero, as in Dupont et al. (2019). Figure 2 shows that our model learns a reasonably flexible approximate posterior on a synthetic non-monotonic 1D dataset. We emphasize that the samples from our model are smooth w.r.t. depth because the hidden states h do not receive additive instantaneous noise, only the weights w do.

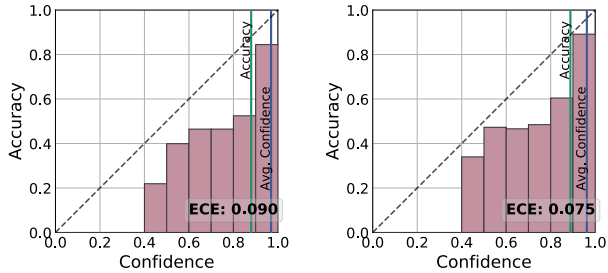
5.2 Image Classification

Instantaneous changes to the hidden state (f_h) are parameterized using a convolutional neural network, including one strided convolution for downsampling and a transposed convolution layer for upsampling. We then set the w to be the filters and biases of all the convolutional layers. The approximate posterior drift dynamics (f_w) is a multilayer perceptron with hidden layer widths of 2, 128, and 2. The small hidden width of the bottleneck layers was chosen to reduce the number of variational parameters and promote linear scaling with respect to the dimension of w . On MNIST, we used one such SDE-BNN block, while on CIFAR-10, we used a multi-scale variant where multiple SDE-BNN blocks were stacked with the invertible downsampling from Dinh et al. (2016) in between.

We report classification results in Table 1. Our SDE-



(a) CIFAR-10. *Left*: Negative log likelihood. *Right*: ECE. Adjusting SDE-BNN solver tolerance at test time trades off computational speed for predictive performance. Grey line is solver’s training tolerance. Averaged across 3 seeds.



(b) Calibration on the CIFAR-10 test set for a neural ODE (left) and a SDE-BNN (right). The SDE-BNN displays better calibration and generalization.

Figure 6: Performance of SDE-BNN on standard CIFAR-10 classification task.

BNN generally outperforms the baselines. While the continuous-depth Neural ODE (ODENet) models can achieve similar classification performance on a standard residual network, it consistently has poorer calibration.

The SDE-BNN matches and outperforms the accuracy of standard residual networks on MNIST and CIFAR-10, respectively, while obtaining lower expected calibration errors (ECE). From ablation studies, we found that it was harder to achieve similar performance with either of the mean field variants of an ODENet as they had a poorer trade-off between performance and calibration.

Figure 6a shows the ability of SDE-BNNs to trade off computation time for precision. Figure 12 in Appendix C.4 indicates that calibration is insensitive to solver tolerances close to the value used during training.

5.2.1 Calibration

Table 1 quantifies our model’s calibration with expected calibration error (ECE; Guo et al. (2017)). The SDE-BNN appears better calibrated than the Neural ODE (Chen et al., 2018) and mean field ResNet baselines. Figure 6b shows better calibration than neural ODEs with similar accuracy. Appendix Figure 11 shows the insensitivity of these results to solver step size.

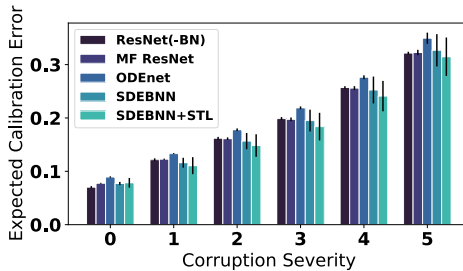
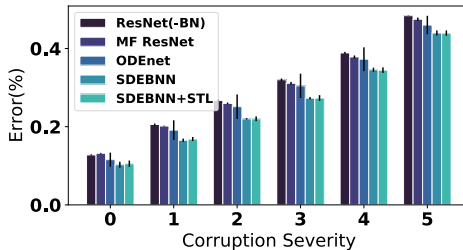


Figure 7: CIFAR10-C. Robustness to distributional shifts on CIFAR-10. SDE-based neural nets show better accuracy and calibration than non-Bayesian and mean-field methods. Black bars show standard deviation over 3 seeds.

5.2.2 Robustness to Input Corruption

We report the robustness of SDE-BNNs by evaluating on all 19 non-adversarial corruptions across 5 severity levels in CIFAR10-C Hendrycks and Dietterich (2019). These corruptions mimic real-world perturbations such as noise, blur, and weather. To evaluate the classification robustness of SDE-BNN, we compare the mean corruption error (mCE), an average error for each intensity level summed across all 19 perturbations, to the top-1 error rate on the corresponding clean CIFAR-10.

Figure 7 shows error on the corrupted test set relative to uncorrupted data, demonstrating a steady increase in mCE across increasing perturbation levels along with the overall error measurement summarized in Table 1. On both CIFAR-10 and CIFAR10-C, the SDE-BNN and SDE-BNN + STL models achieve lower overall test error and better calibration than the baselines.

Compared to standard baselines (ResNet32 and Mean Field (MF) ResNet32), SDE-BNN achieves around 4.4% lower absolute corruption error (CE), the total classification error for all corruption tasks across all 5 severity levels (Hendrycks and Dietterich, 2019), in comparison to the clean errors. The effectiveness of learned uncertainty on out-of-domain inputs indicates that SDE-BNN is more robust to observation noise despite not being trained on such diverse forms of corruptions.

6 SCOPE AND LIMITATIONS

Computational speed The cost of evaluating our model grows in $\mathcal{O}(DT)$, where D is the number of weights, and T the number of iterations taken by the solver. This may seem advantageous compared to the $\mathcal{O}(D^3)$ cost for non-factorized Gaussian approximate posteriors, but the number of steps required is difficult to characterize. Although our approach allows adjustment of the computational cost at test time, it is harder to control the cost of evaluation during training time, making our method relatively slow to train. However, it should be straightforward to regularize these models to be faster to solve, as in Kelly et al. (2020). Relatedly, Dusenberry et al. (2020) recently demonstrated an $\mathcal{O}(DK)$ cost approximate posterior in standard BNNs.

Batch norm We did not incorporate batch normalization (Ioffe and Szegedy, 2015) in any of our neural network components. Introducing any normalization (e.g. batchnorm, layernorm, etc.) compromises the Lipschitz property required for SDEs to have a unique solution. Since BN introduces dependence between samples within a batch, it is also unclear how to incorporate BN while maintaining the consistency properties of Bayesian inference. Zhang et al. (2019a); Chang et al. (2020) proposed initializations that yield the same performance without needing batch normalization.

Low-variance gradients for other domains Our extended STL gradient estimator (Roeder et al., 2017) to the infinite-dimensional variational objective could be applied to other settings for faster convergence, e.g. time series applications Li et al. (2020) investigated.

7 RELATED WORK

Initial theoretical investigations The earliest theoretical treatment of infinitely-deep Bayesian neural networks was made by Neal (1996, Chapter 2), but no practical training or evaluation method was proposed. Duvenaud et al. (2014) also investigated the theoretical properties of kernel-based constructions of infinitely-deep Bayesian neural networks.

Diffusion limits of discrete-time models We expect existing discrete-depth constructions to converge to diffusion limits in the infinitesimal limit if a system is updated with appropriately scaled Gaussian noise at each timestep. Peluchetti and Favaro (2020b,a) show this holds for the output of residual networks with shallow residual blocks whose weight initializations are appropriately scaled. While our construction of SDE-BNN given by equation 11 seems similar Peluchetti and Favaro (2020b), there are two key differences: (i)

We strictly enforce hidden states to follow a diffusion throughout training by directly learning a neural SDE, whereas Peluchetti and Favaro (2020b) only ensures SDE-driven dynamics at initialization. (ii) We adopt a more general neural net architecture for the residual blocks than the shallow ones considered in (Peluchetti and Favaro, 2020b). Their work mainly discusses the convergence of shallow ResNets to SDEs, in order to analyze training stability for regular ResNets and verify that a scaled gradient formulation leads to faster convergence at the first epoch. The consequence of (i) is that operations on diffusions (e.g., computing path-space KL) remain applicable even after our model has been trained. While (ii) appears to be a minor difference, it actually uncovers a fundamental distinction in our analysis: Since we start out with an SDE, and only discretize for numerical computations, our model is able to incorporate any type of Lipschitz smooth residual block. Additionally, no training algorithm was specified for learning SDE models. The analysis by Peluchetti and Favaro (2020b) relies on Taylor expanding the residual block function, which is not easy in the presence of complex residual block architectures and would require modifications to the initialization. Tzen and Raginsky (2019a) show that particle trajectories of the approximate posterior in discrete deep latent Gaussian models converge to a diffusion, and that the ELBO may be written with KL of measures on path space. This construction has been explored in various forms in the past (Oppen, 2019; Archambeau et al., 2008).

Neural SDEs with other training objectives Models making use of SDEs have appeared in the past, though many make use of somewhat *ad-hoc* combinations of methods involving both discrete and continuous components. Kong et al. (2020) proposed fitting a neural SDE by using a heuristic training objective based on encouraging the diffusion to be large away from the training data and a fixed Euler-Maruyama (E-M) discretization. Innes et al. (2019) trained neural SDEs by backpropagating through the operations of the solver, however their training objective simply matched the first two moments of the training data, implying that it could not consistently estimate diffusion functions. This approach is also relatively memory-intensive. Liu et al. (2019) and Oganessian et al. (2020) add noise to the solver operations in a neural ODE, although the diffusion must be tuned as a hyperparameter. Hegde et al. (2018) proposed a form of neural SDE using Gaussian processes to parameterize the drift and diffusion functions for a fixed E-M discretization. However, the diffusion functions are based on an *ad-hoc* construction from a Gaussian process posterior conditioned on inducing points. Ryder et al. (2018) used a Gaussian

process variational posterior, effectively a continuous-time analog of a mean field approximation that may not always be expressive enough to model the true posterior. Kidger et al. (2021) learn neural SDEs by jointly learning a discriminator (Kidger et al., 2020) and formalize the problem as learning generative adversarial networks. However, this would involve many more hyperparameters and require extensive tuning compared to our variational inference approach.

ODEnets with finite-dimensional stochasticity

Some methods based on building variational autoencoders with a neural ODE share similar training objectives, since the ELBO appears frequently in posterior inference. The Latent ODE model (Rubanova et al., 2019) only performs inference on the distribution at an initial time of a continuous hidden state. De Brouwer et al. (2019) introduced stochastic jumps at data locations, and do not perform continuous-time inference. While performing amortized inference for time series modeling, Yıldız et al. (2019) also infer the weights of an ODE drift function. Dandekar et al. (2020) have a similar setting but for supervised learning.

Approximate posteriors defined as neural nets

Krueger et al. (2018) and Louizos and Welling (2017) use normalizing flows to construct an unfactorized, non-Gaussian approximate posterior in BNNs. However, normalizing flows have poor scaling with dimension and point estimates were used for most of the weights in the neural network. Table 5 in Appendix 5 compares qualities of our approach to existing methods for stochastic variational inference in BNNs.

8 CONCLUSION

We developed a practical method for approximate inference in continuous-depth Bayesian neural networks. Our approach exploits a special synergy between continuous-depth models and variational inference for SDEs, providing additional benefits over standard approaches. In particular, our method allows arbitrarily-expressive, non-factorized approximate posteriors implicitly defined through neural SDEs. We also developed an unbiased gradient estimator for SDE variational inference whose variance approaches zero as the approximate posterior approaches the true posterior. This combination gives our family of Bayesian continuous-depth neural networks a special property, which is that the gradients’ bias and variance can be made arbitrarily small during training. Where standard applications of MFVI on continuous-depth models perform poorly, our approach brings continuous-depth Bayesian neural networks to a comparable performance with standard Bayesian neural networks. Furthermore,

we demonstrated the ability of this continuous-depth model class to use adaptive SDE solvers. This allows a memory-efficient training, and a fine-grained trade-off between precision and speed.

Acknowledgements

We thank Jesse Bettencourt, Radford M. Neal, and Patrick Kidger for helpful technical discussions and revisions on earlier drafts of this work. We also thank James Bradbury for his support while implementing differential equation solvers in JAX.

References

- Archambeau, C., Opper, M., Shen, Y., Cornford, D., and Shawe-Taylor, J. (2008). Variational inference for diffusion processes. *Advances in Neural Information Processing Systems*.
- Balan, A. K., Rathod, V., Murphy, K. P., and Welling, M. (2015). Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*, pages 3438–3446.
- Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *International Conference on Machine Learning*.
- Boué, M., Dupuis, P., et al. (1998). A variational representation for certain functionals of brownian motion. *The Annals of Probability*, 26(4):1641–1659.
- Chang, O., Flokas, L., and Lipson, H. (2020). Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*.
- Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. *Advances in Neural Information Processing Systems*.
- Dandekar, R., Dixit, V., Tarek, M., Garcia-Valadez, A., and Rackauckas, C. (2020). Bayesian neural ordinary differential equations. *arXiv preprint arXiv:2012.07244*.
- De Brouwer, E., Simm, J., Arany, A., and Moreau, Y. (2019). Gru-ode-bayes: Continuous modeling of sporadically-observed time series. *arXiv preprint arXiv:1905.12374*.
- Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.
- Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

- Dupont, E., Doucet, A., and Teh, Y. W. (2019). Augmented neural odes. *Advances in Neural Information Processing Systems*.
- Dusenberry, M., Jerfel, G., Wen, Y., Ma, Y., Snoek, J., Heller, K., Lakshminarayanan, B., and Tran, D. (2020). Efficient and scalable bayesian neural nets with rank-1 factors. In *International conference on machine learning*, pages 2782–2792. PMLR.
- Duvenaud, D., Rippel, O., Adams, R. P., and Ghahramani, Z. (2014). Avoiding pathologies in very deep networks. In *Artificial Intelligence and Statistics*.
- E, W. (2017). A Proposal on Machine Learning via Dynamical Systems. *Commun. Math. Stat.*, 5(1):1–11.
- Graves, A. (2011). Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356.
- Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org.
- Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.
- Haber, E. and Ruthotto, L. (2017). Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004.
- Hegde, P., Heinonen, M., Lähdesmäki, H., and Kaski, S. (2018). Deep learning with differential gaussian process flows. *arXiv preprint arXiv:1810.04066*.
- Hendrycks, D. and Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*.
- Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.
- Innes, M., Edelman, A., Fischer, K., Rackauckus, C., Saba, E., Shah, V. B., and Tebbutt, W. (2019). Zygote: A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587*, page 140.
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.
- Izmailov, P., Vikram, S., Hoffman, M. D., and Wilson, A. G. (2021). What are bayesian neural network posteriors really like? *International Conference on Learning Representations*.
- Kelly, J., Bettencourt, J., Johnson, M. J., and Duvenaud, D. (2020). Learning differential equations that are easy to solve. In *Neural Information Processing Systems*.
- Kidger, P., Foster, J., Li, X., Oberhauser, H., and Lyons, T. (2021). Neural sdes as infinite-dimensional gans. *arXiv preprint arXiv:2102.03657*.
- Kidger, P., Morrill, J., Foster, J., and Lyons, T. (2020). Neural controlled differential equations for irregular time series. *arXiv preprint arXiv:2005.08926*.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.
- Kong, L., Sun, J., and Zhang, C. (2020). Sde-net: Equipping deep neural networks with uncertainty estimates. *arXiv preprint arXiv:2008.10546*.
- Krizhevsky, A., Nair, V., and Hinton, G. (2014). The cifar-10 dataset. *online: http://www.cs.toronto.edu/kriz/cifar.html*, 55:5.
- Krueger, D., Huang, C.-W., Islam, R., Turner, R., Lacoste, A., and Courville, A. (2018). Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*.
- Li, X., Wong, T.-K. L., Chen, R. T., and Duvenaud, D. (2020). Scalable gradients for stochastic differential equations. *arXiv preprint arXiv:2001.01328*.
- Liu, X., Xiao, T., Si, S., Cao, Q., Kumar, S., and Hsieh, C.-J. (2019). Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*.
- Look, A. and Kandemir, M. (2019). Differential bayesian neural nets. *arXiv preprint arXiv:1912.00796*.
- Louizos, C. and Welling, M. (2017). Multiplicative normalizing flows for variational bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2218–2227. JMLR. org.
- Mishkin, A., Kunstner, F., Nielsen, D., Schmidt, M., and Khan, M. E. (2018). Slang: Fast structured covariance approximations for bayesian deep learning with natural gradient. In *Advances in Neural Information Processing Systems*, pages 6245–6255.
- Neal, R. M. (1996). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.
- Oganesyan, V., Volokhova, A., and Vetrov, D. (2020). Stochasticity in neural odes: An empirical study. *arXiv preprint arXiv:2002.09779*.
- Opper, M. (2019). Variational inference for stochastic differential equations. *Annalen der Physik*, 531(3):1800233.

- Osawa, K., Swaroop, S., Jain, A., Eschenhagen, R., Turner, R. E., Yokota, R., and Khan, M. E. (2019). Practical deep learning with bayesian principles. *arXiv preprint arXiv:1906.02506*.
- Peluchetti, S. and Favaro, S. (2020a). Doubly infinite residual networks: a diffusion process approach.
- Peluchetti, S. and Favaro, S. (2020b). Infinitely deep neural networks as diffusion processes. In *International Conference on Artificial Intelligence and Statistics*, pages 1126–1136. PMLR.
- Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.
- Roeder, G., Wu, Y., and Duvenaud, D. K. (2017). Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In *Advances in Neural Information Processing Systems*.
- Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. (2019). Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*.
- Ryder, T., Golightly, A., McGough, A. S., and Prangle, D. (2018). Black-box variational inference for stochastic differential equations. In *International Conference on Machine Learning*.
- Tzen, B. and Raginsky, M. (2019a). Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*.
- Tzen, B. and Raginsky, M. (2019b). Theoretical guarantees for sampling and inference in generative models with latent diffusions. *arXiv preprint arXiv:1903.01608*.
- Wang, K.-C., Vicol, P., Lucas, J., Gu, L., Grosse, R., and Zemel, R. (2018). Adversarial distillation of bayesian neural network posteriors. *arXiv preprint arXiv:1806.10317*.
- Wenzel, F., Roth, K., Veeling, B., Swiatkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. (2020). How good is the bayes posterior in deep neural networks really? In *International Conference on Machine Learning*.
- Yıldız, Ç., Heinonen, M., and Lähdesmäki, H. (2019). Ode²vae: Deep generative second order odes with bayesian neural networks. *arXiv preprint arXiv:1905.10994*.
- Zadrozny, B. and Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. (2018). Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, pages 5852–5861.
- Zhang, H., Dauphin, Y. N., and Ma, T. (2019a). Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*.
- Zhang, H., Gao, X., Unterman, J., and Arodz, T. (2019b). Approximation capabilities of neural ordinary differential equations. *arXiv preprint arXiv:1907.12998*.
- Zhang, T., Yao, Z., Gholami, A., Keutzer, K., Gonzalez, J., Biro, G., and Mahoney, M. W. (2019c). ANODEV2: A coupled neural ODE evolution framework. *CoRR*, abs/1906.04596.

Supplementary Material: Infinitely Deep Bayesian Neural Networks with Stochastic Differential Equations

A PROOFS

Notation. Denote as ϕ the vector of variational parameters, f_q as the approximate posterior on weights, f_p as the prior on weights, f_h as the dynamics of hidden units, and σ as the diffusion function. Denote the Euclidean norm of a vector u by $|u|$. For function f denote its Jacobian as ∇f .

A.1 Derivation of an Alternative Monte Carlo Estimator

The goal of this section is to derive a Monte Carlo estimator of the KL-divergence on path space that is similar to the *fully Monte Carlo* estimator described in Roeder et al. (2017). This will serve as the basis for the subsequent heuristic derivation of the continuous-time sticking-the-landing trick.

Let w_0 be a fixed initial state. Let w_1, \dots, w_N be states at times $\Delta t, 2\Delta t, \dots, N\Delta t = T$ generated by the Euler discretization:

$$w_{i+1} = w_i + f_q(w_i)\Delta t + \sigma(w_i)(B_{t+\Delta t} - B_t) \quad (13)$$

$$= w_i + f_q(w_i)\Delta t + \sigma(w_i)\Delta t^{1/2}\epsilon_{i+1}, \quad \epsilon_{i+1} \sim \mathcal{N}(0, 1). \quad (14)$$

where $\{B_t\}_{t \geq 0}$ is the Brownian motion. This implies that conditional on the previous state, the current state is normally distributed:

$$w_{i+1}|w_i \sim \mathcal{N}(w_i + f_q(w_i)\Delta t, \sigma(w_i)^2\Delta t).$$

Thus, the log-densities can be evaluated as

$$\log q(w_{i+1}|w_i) = -\frac{1}{2} \log(2\pi\sigma(w_i)^2\Delta t) - \frac{1}{2} \frac{(w_{i+1} - (w_i + f_q(w_i)\Delta t))^2}{\sigma(w_i)^2\Delta t}, \quad i = 0, \dots, N-1. \quad (15)$$

On the other hand, if at any time, the next state was generated from the current state based on the prior process, we would have the following log-densities:

$$\log p(w_{i+1}|w_i) = -\frac{1}{2} \log(2\pi\sigma(w_i)^2\Delta t) - \frac{1}{2} \frac{(w_{i+1} - (w_i + f_p(w_i)\Delta t))^2}{\sigma(w_i)^2\Delta t}, \quad i = 0, \dots, N-1. \quad (16)$$

Now, we substitute the form of w_{i+1} based on equation 13 into equation 15 and equation 16 and obtain

$$\begin{aligned} \log q(w_{i+1}|w_i) &= -\frac{1}{2} \log(2\pi\sigma(w_i)^2\Delta t) - \frac{1}{2} \epsilon_{i+1}^2, \\ \log p(w_{i+1}|w_i) &= -\frac{1}{2} \log(2\pi\sigma(w_i)^2\Delta t) \\ &\quad - \frac{1}{2} \left(\frac{(f_q(w_i) - f_p(w_i))^2}{\sigma(w_i)^2} \Delta t + \frac{2(f_q(w_i) - f_p(w_i))\epsilon_{i+1}}{\sigma(w_i)} \Delta t^{1/2} + \epsilon_{i+1}^2 \right). \end{aligned}$$

The KL divergence on the path space could then be regarded as a sum of infinitely many KL-divergences between

Gaussians:

$$\lim_{N \rightarrow \infty} \sum_{i=0}^N \mathbb{E}_{w_i} [D_{\text{KL}}(q(w_{i+1}|w_i)||p(w_{i+1}|w_i))] \quad (17)$$

$$= \lim_{N \rightarrow \infty} \sum_{i=0}^N \mathbb{E}_{w_i} \left[\mathbb{E}_{w_{i+1} \sim q(w_{i+1}|w_i)} \left[\log \frac{q(w_{i+1}|w_i)}{p(w_{i+1}|w_i)} \right] \right] \quad (18)$$

$$= \lim_{N \rightarrow \infty} \sum_{i=0}^N \mathbb{E}_{w_i} \left[\mathbb{E}_{\epsilon_{i+1}} \left[\frac{(f_q(w_i) - f_p(w_i))^2}{2\sigma(w_i)^2} \Delta t + \frac{(f_q(w_i) - f_p(w_i))}{\sigma(w_i)} \Delta t^{1/2} \epsilon_{i+1} \right] \right] \quad (19)$$

$$= \mathbb{E} \left[\frac{1}{2} \int_0^T |u_t|^2 dt + \int_0^T u_t dB_t \right]. \quad (20)$$

A.2 Sticking-the-landing in Continuous Time

For a non-sequential latent variable model, the sticking-the-landing (STL) trick removes from the fully Monte Carlo ELBO estimator a score function term of the form $\partial \log q(w, \phi) / \partial \phi$, where w is sampled using the reparameterization trick and may depend on ϕ . The score function term has 0 expectation, but may affect the variance of the gradient estimator for the inference distribution's parameters.

Here, we exploit this intuition and apply it to each step before taking the limit. More precisely, we apply the STL trick to estimate the gradient of $D_{\text{KL}}(q(w_{i+1}|w_i)||p(w_{i+1}|w_i))$ for $i = 1, 2, \dots, N$, and thereafter take the limit as the mesh size of the discretization goes to 0. For each individual term, the score function term to be removed is

$$\begin{aligned} \frac{\partial}{\partial \phi} \log q(w_{i+1}|w_i, \phi) &= - \frac{1}{2\sigma^2(w_i)\Delta t} \frac{\partial}{\partial \phi} \left[(w_{i+1} - (w_i + f_q(w_i, \phi)\Delta t))^2 \right] \\ &= \frac{\partial}{\partial \phi} \left[\frac{f_q(w_i, \phi)}{\sigma(w_i)} \right] \epsilon_{i+1} \Delta t^{1/2}. \end{aligned}$$

Now, we sum up all of these terms and take the limit as $\Delta t \rightarrow 0$. This gives us

$$\begin{aligned} &\lim_{N \rightarrow \infty} \sum_{i=0}^N \mathbb{E}_{w_i} \left[\mathbb{E}_{w_{i+1} \sim q(w_{i+1}|w_i)} \left[\frac{\partial}{\partial \phi} \log q(w_{i+1}|w_i) \right] \right] \\ &= \lim_{N \rightarrow \infty} \sum_{i=0}^N \mathbb{E}_{w_i} \left[\mathbb{E}_{\epsilon_{i+1}} \left[\frac{\partial}{\partial \phi} \left[\frac{f_q(w_i, \phi)}{\sigma(w_i)} \right] \epsilon_{i+1} \Delta t^{1/2} \right] \right] \\ &= \mathbb{E} \left[\int_0^T \frac{\partial}{\partial \phi} \left[\frac{f_q(w_t, \phi)}{\sigma(w_t)} \right] dB_t \right] \\ &= \mathbb{E} \left[\int_0^T \frac{\partial}{\partial \phi} [u_t] dB_t \right]. \end{aligned}$$

Removing this term from the fully Monte Carlo estimator in equation 20 gives rise to the following estimator of a surrogate objective that facilitates implementation:

$$\begin{aligned} \widehat{\text{ELBO}} &= \log p(\mathcal{D} | w) - \int_{t_0}^{t_1} \frac{1}{2} \|u(w_t, t, \phi)\|_2^2 dt \\ &\quad - \int_{t_0}^{t_1} u(w_t, t, \text{stop_gradient}(\phi)) dB_t, \quad w(\cdot) \sim q_\phi(\cdot). \end{aligned}$$

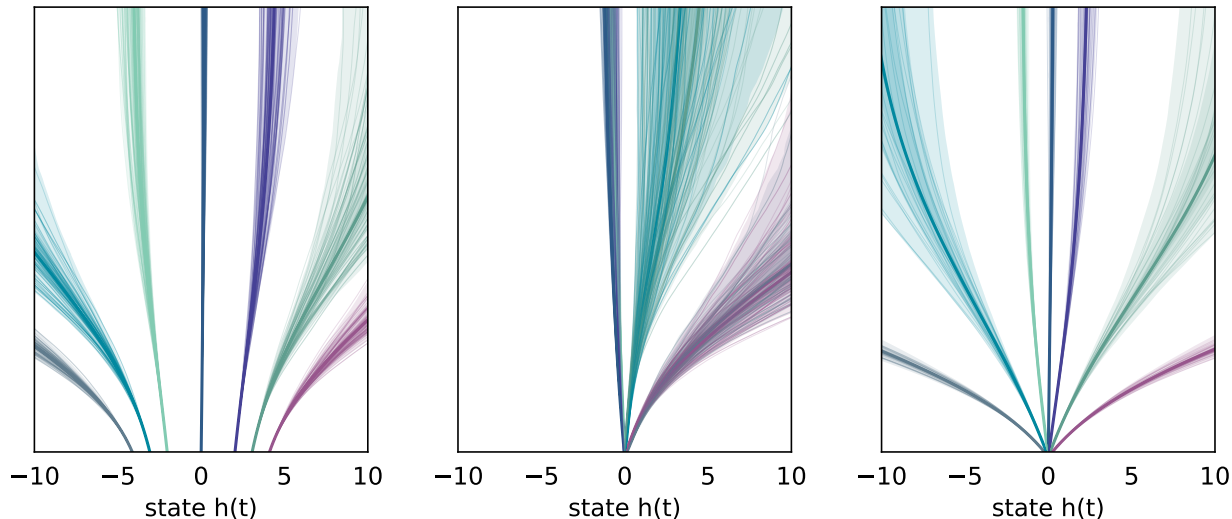
B EXPERIMENTAL SETTINGS

Table 2: These are the hyper-parameters for each method of evaluation pertaining to results in the toy and classification tasks of Table 1. Each model was run on a single Nvidia RTX6000 GPU on our compute clusters. SDE and learning optimization parameters were tuned according to a validation set sampled randomly from 10% of the training set. No schedules of any kind on the hyper-parameters were used in training. Settings with high overlap with another model are indicated using `<model>` with additional parameters overridden as necessary. Each block is separated either by a downsampling or upsampling convolutional layer (i.e., the -'s).

Model	Hyper-parameter	EXPERIMENTS		
		1D Regression	MNIST Deng (2012)	CIFAR-10 Krizhevsky et al. (2014)
ResNet32	Learning Rate	–	1e-3	7e-4
	Batch Size	–	128	128
	Activation	–	tanh	tanh
	Epochs	–	100	500
ODEnet	Augment dim.	2	2	2
	# blocks	1	1	2-2-2
	Diffusion σ	0	0	0
	KL coef.	0	0	0
	Learning Rate	1e-3	1e-3	7e-4
	# Solver Steps	10	20	20
	Batch Size	40	128	128
	Activation	swish	tanh	tanh
Epochs	800	100	500	
HyperODEnet	<ODEnet>	–	<ODEnet>	<ODEnet>
	KL coef.	–	1e-3	1e-3
	Drift f_w dim.	–	1-64-1	1-128-1
MFVI ResNet32	<ResNet32>	–	<ResNet32>	<ResNet32>
	KL coef.	–	1e-3	1e-3
MFVI ODEnet	<ODEnet>	–	<ODEnet>	<ODEnet>
	KL coef.	–	1e-3	1e-3
MFVI HyperODEnet	<MFVI ODEnet>	–	<MFVI ODEnet>	<MFVI ODEnet>
	Drift f_w dim.	–	1-64-1	1-128-1
SDE BNN	<ODEnet>	<ODEnet>	<ODEnet>	<ODEnet>
	Learning Rate	1e-3	1e-3	7e-4
	# blocks	1	1	2-2-2
	Drift f_x dim.	32	32	64
	Drift f_w dim.	32	1-64-1	2-128-2
	Diffusion σ	0.2	0.1	0.1
	# Posterior Samples	20	1	1
SDE BNN (+ STL)	<SDE BNN>	<SDE BNN>	<SDE BNN>	<SDE BNN>

C ADDITIONAL RESULTS

C.1 Augmentation in Differential Equation Models



(a) Non-augmented dimension (b) from 2nd augmented dimension (c) from last augmented dimension

Figure 8: Example flows sampled from learned SDE dynamics. All continuous-depth models were trained by augmenting the state by 2 dimensions, refer to Figure 1 for main results. *Left*: The SDE-BNN learns meaningful parameterizations on the non-extraneous dimensions of the input state vector. In the case of a true function being monotonic, the augmented dimensions simply help the main output. *Middle*: The model learns to ignore dimensions that are not necessary to train on, especially on simpler tasks as in the toy setting. Samples in augmented dimensions can overlap for different input values in the given domain $(-5, 5)$. *Right*: Similarly, the last output dimension was also associated with augmentation and was not a well learned representation of the data, ignoring the initial inputs entirely (all values are 0).

C.2 Classification Results

Table 3: Classification accuracy and expected calibration error on MNIST (100th epoch) and CIFAR-10 (300th epoch) for additional baseline to the ones in Table 1. Values are compared at the 100th epoch for MNIST and 300th for CIFAR-10. Here w_0 , the initial drift of the posterior SDE, is inferred using a Gaussian prior rather than being a fixed value. The best prior variance was selected in a preliminary sweep between values in the range $[0.1, 0.44]$. The performance is slightly worse than the point estimate but displays better calibration as a trade-off. It can be noted that calibration may appear better earlier on in training, as in prior to converging and reaching non-uniform confidence, but model predictions are not necessarily correct.

Model	MNIST		CIFAR-10	
	Accuracy (%)	ECE ($\times 10^{-2}$)	Accuracy (%)	ECE ($\times 10^{-2}$)
SDE BNN	99.30 \pm 0.09	0.63 \pm 0.10	88.08 \pm 1.25	7.53 \pm 0.44
SDE BNN (+ STL)	99.10 \pm 0.09	0.78 \pm 0.12	87.95 \pm 1.32	7.94 \pm 0.59
SDE BNN w_0 inferred	99.04 \pm 0.03	0.73 \pm 0.04	88.04 \pm 0.30	6.56 \pm 0.62
SDE BNN w_0 inferred (+ STL)	99.05 \pm 0.00	0.79 \pm 0.04	87.37 \pm 0.63	6.44 \pm 0.10

C.3 Sticking the Landing Results

Table 4: Training with STL estimator on CIFAR-10 shows training time improvements in accuracy, negative log likelihood, and ELBO objective in addition to reducing variance. This improvement to the standard gradient estimator can be especially useful in settings where the approximate posterior is sufficiently flexible (i.e. the drift neural net is very large relative to the state size).

Method	Accuracy (%)	Negative Log-likelihood ($\times 10^{-4}$)	ELBO
SDE BNN	95.91 \pm 0.2	1.17 \pm 0.309	1.40 \pm 0.2
SDE BNN (+STL)	96.89 \pm 0.2	0.309 \pm 0.15	1.183 \pm 0.2

C.4 Calibration Results

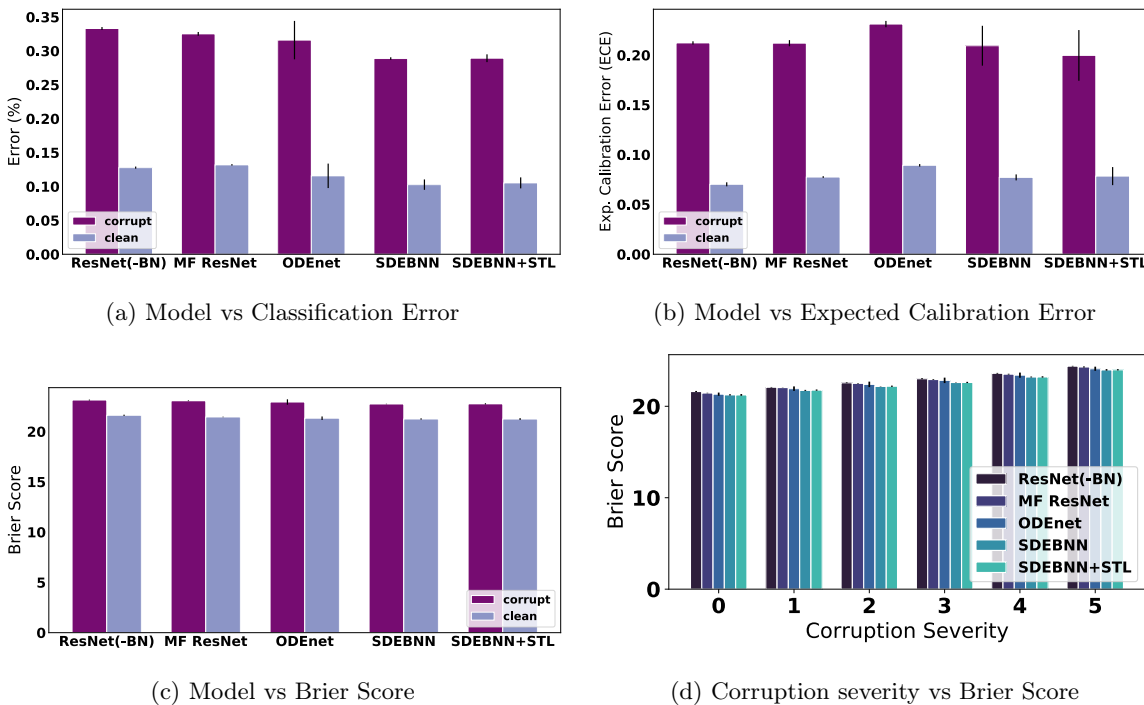


Figure 9: Figures 9a-9c show that the SDE BNN and SDE BNN + STL models outperform their non-continuous depth ResNet counterparts on all three robustness metrics when evaluated on the corrupt CIFAR-10C benchmarks. Figure 9d indicates that the accuracy of predictions is relatively consistent across all severity levels with the SDE-BNN and SDE-BNN + STL models having relatively better calibrated predictions.

C.5 Comparisons with Other Bayesian Models

Table 5: Properties of various Bayesian supervised learning approaches.

Method	Posterior over Stochastic Process	Flexible Approximate Posterior	Adaptive Computation	References
Bayes by Backprop	✗	✗	✗	Blundell et al. (2015)
MCMC for BNNs	✗	✓	✗	(Neal, 1996; Wenzel et al., 2020; Izmailov et al., 2021)
Bayesian Hypernets	✗	✓	✗	Krueger et al. (2018)
BBVI for SDEs	✓	✗	✗	Ryder et al. (2018)
Bayesian Neural ODEs	✗	✗	✓	Yildiz et al. (2019) Dandekar et al. (2020)
SDE-BNN	✓	✓	✓	current work

C.6 Other Bayesian Methods

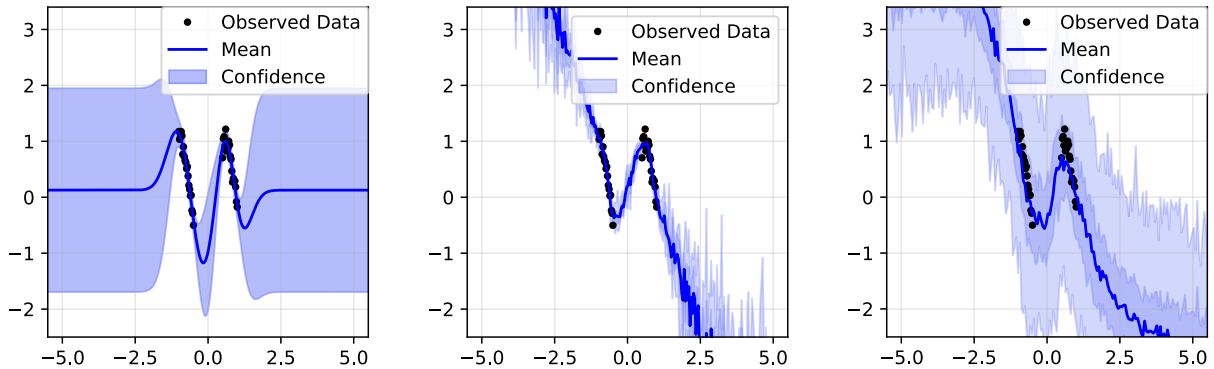


Figure 10: Approximate posteriors from other common Bayesian statistical models. *Left*: Gaussian Process. *Center*: Deep Ensemble $K=8$. *Right*: MFVI. Different variances and extrapolations are learned from the SDE-BNN across other Bayesian model parameterizations, which can result in more or less reasonable uncertainty bounds depending on interpretation.

C.7 Robustness to solver error at test time

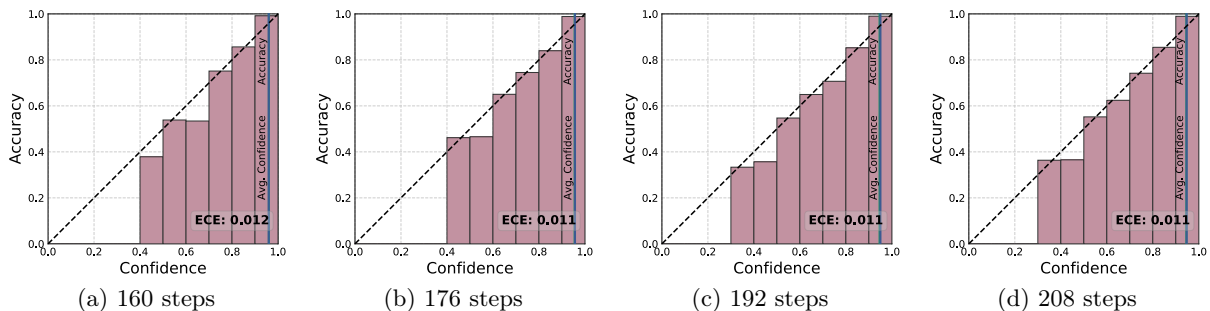


Figure 11: CIFAR10 image classification with a SDE-BNN. Better calibration can be obtained by increasing solver step sizes during inference without substantially changing the training error.

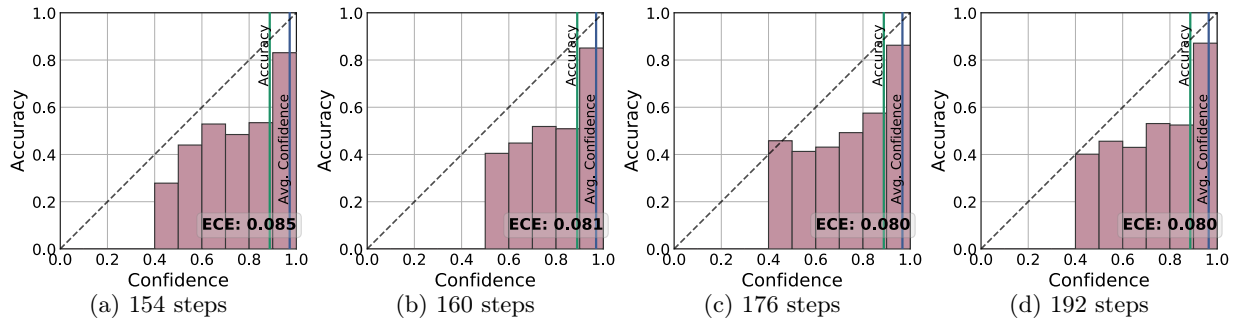


Figure 12: CIFAR10 image classification with a SDE-BNN. Generalization improves marginally compared to a trained model during inference in 12b, as tuning solver step size does not yield significant differences in calibration outcomes.

C.8 SDE solver and adjoint settings

These were run with a SDE-BNN for MNIST image classification, to compare the performance and run-time cost across different solver settings. Comparably, backpropagation through the solver averaged 162.58 sec / epoch while the adjoint method averaged 135.90 sec / epoch in terms of wall clock time.

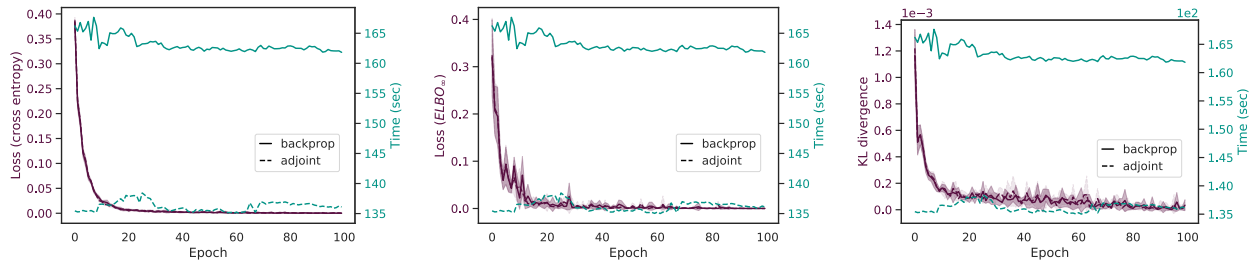


Figure 13: Backpropagation through the SDE solver yields similar optimization dynamics but is less time efficient than the adjoint method.

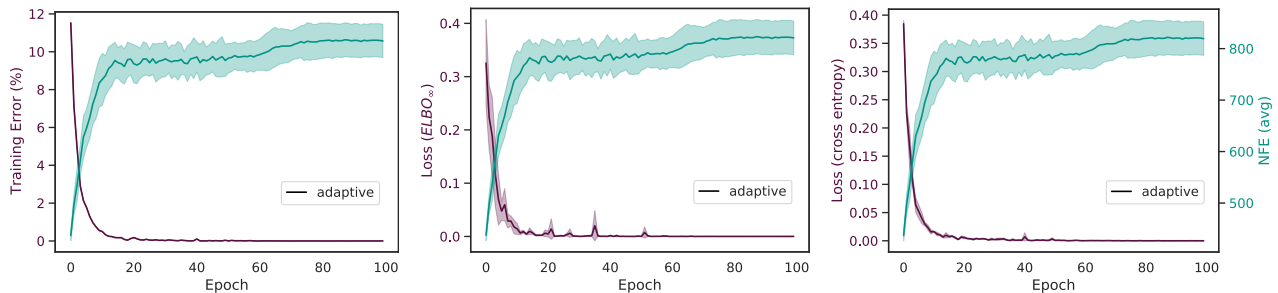


Figure 14: Trade-off between solver speed and convergence during training. Adaptive refers to training with the stochastic adjoint in both forward and reverse modes here.

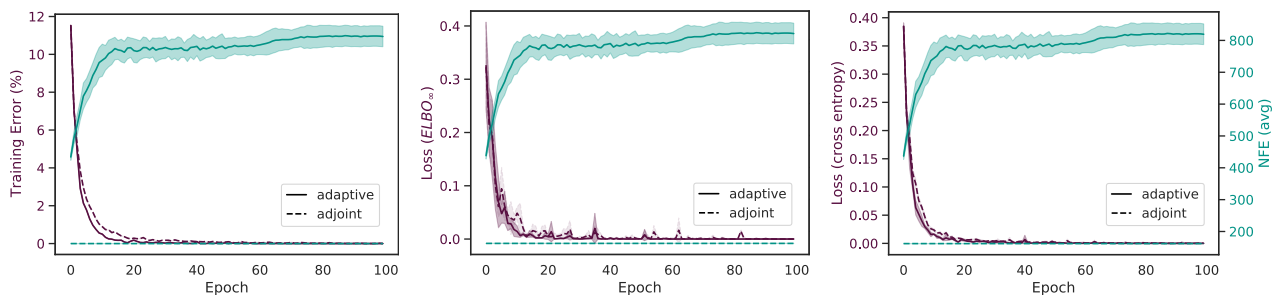


Figure 15: Trade-off between solver speed and precision during training. Adaptive-order optimization trajectories were comparable to fixed-order solvers and were thus not applied to the classification tasks since computational resources were not under constraint.