ON THE PITFALLS OF HETEROSCEDASTIC UNCERTAINTY ESTIMATION WITH PROBABILISTIC NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Capturing aleatoric uncertainty is a critical part of many machine learning systems. In deep learning, a common approach to this end is to train a neural network to estimate the parameters of a heteroscedastic Gaussian distribution by maximizing the logarithm of the likelihood function under the observed data. In this work, we examine this approach and identify potential hazards associated with the use of log-likelihood in conjunction with gradient-based optimizers. First, we present a synthetic example illustrating how this approach can lead to very poor but stable parameter estimates. Second, we identify the culprit to be the log-likelihood loss, along with certain conditions that exacerbate the issue. Third, we present an alternative formulation in which each data point's contribution to the loss is weighted by the β-exponentiated variance estimate. We show that using an appropriate β largely mitigates the issue in our illustrative example. Fourth, we evaluate this approach on a range of standard benchmarks from the literature and show that it achieves considerable improvements and performs more robustly with respect to hyperparameters, both in predictive RMSE and log-likelihood criteria.

1 INTRODUCTION

Endowing models with the ability to capture *uncertainty* is of crucial importance in machine learning. Uncertainty can be usefully categorized into two main types: *epistemic* uncertainty and *aleatoric* uncertainty (Kiureghian & Ditlevsen, 2009). Epistemic uncertainty accounts for subjective uncertainty in the model, one that is reducible given sufficient data. By contrast, aleatoric uncertainty captures the stochasticity inherent in the observations and can itself be subdivided into *homoscedastic* and *heteroscedastic* uncertainty. Homoscedastic uncertainty is noise that stays constant across different inputs, whereas heteroscedastic uncertainty is one that varies depending on the inputs to the model.

There are well-established benefits for modeling each type of uncertainty. For instance, capturing epistemic uncertainty enables effective budgeted data collection in active learning (Gal et al., 2017), allows for efficient exploration in reinforcement learning (Osband et al., 2016), and is indispensable in cost-sensitive decision making (Amodei et al., 2016). On the other hand, quantifying aleatoric uncertainty enables learning dynamics models of stochastic processes (e.g. for model-based or offline reinforcement learning) (Chua et al., 2018; Yu et al., 2020), improves performance in semantic segmentation, depth regression and object detection (Kendall & Gal, 2017; Harakeh & Waslander, 2021), and allows for risk-sensitive decision making (Dabney et al., 2018; Vlastelica et al., 2021).

Many modern applications of machine learning rely on neural networks and deep learning tools to achieve state-of-the-art performance. However, most neural network models are not readily equipped with a capacity for uncertainty estimation. To address this shortcoming, a multitude of approaches have been proposed in the past few years. Nevertheless, the majority of work in this domain has focused on epistemic uncertainty quantification (Blundell et al., 2015; Gal & Ghahramani, 2016) or uncertainty estimation for classification (Hendrycks & Gimpel, 2017; Guo et al., 2017; Ovadia et al., 2019; Mukhoti et al., 2021).

We examine a common approach for quantifying aleatoric uncertainty in neural network regression. By assuming that the regression targets follow a particular distribution, we can use a neural network to predict the parameters of that distribution, typically the input-dependent mean and variance when assuming a heteroscedastic Gaussian distribution. Then, the parameters of the network can be learned using maximum likelihood estimation (MLE), i.e. by minimizing the negative log-likelihood (NLL)

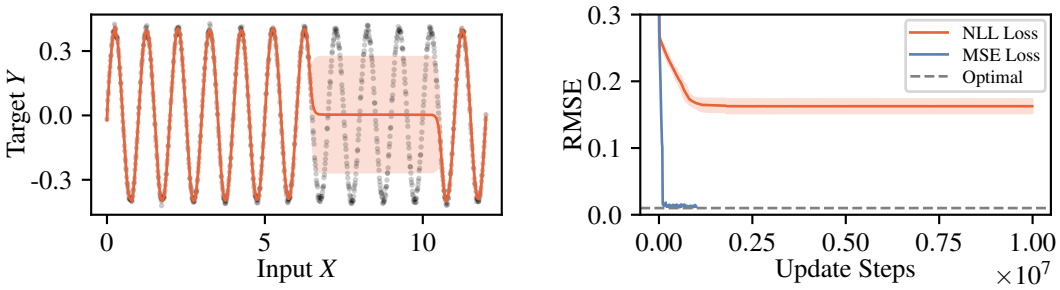

Figure 1: Training a probabilistic neural network to fit a simple sinusoidal fails. Left: learned predictions (orange line) after 10^7 updates, with the shaded region showing the predicted standard deviation. The target function is given by $y(x) = 0.4\sin(2\pi x) + \xi$, where ξ is Gaussian noise with a standard deviation of 0.01. Right: root mean squared error (RMSE) over training, mean and standard deviation over 10 random seeds. For comparison, we plot the training curve when using the mean squared error as the training objective — achieving an optimal mean fit (dashed line) in 10^5 updates.

criterion using stochastic gradient descent. This simple procedure, which is the de-facto standard (Nix & Weigend, 1994; Lakshminarayanan et al., 2017; Kendall & Gal, 2017; Chua et al., 2018; Kloss et al., 2021), is known to be subject to overconfident variance estimates. Whereas strategies have been proposed to alleviate this specific issue (Detlefsen et al., 2019; Hu et al., 2020; Stirn & Knowles, 2020), we argue that an equally important issue is that this procedure can additionally lead to subpar mean fits. In this work, we analyse and propose a simple modification to mitigate this issue.

Summary of contributions We demonstrate a pitfall of optimizing the NLL loss for neural network regression, one that hinders the training of accurate mean predictors (see Fig. 1 for an illustrative example). The primary culprit is the *high dependence of the gradients on the predictive variance*. While such dependence is generally known to be responsible for instabilities in joint optimization of mean and variance estimators (Takahashi et al., 2018; Stirn & Knowles, 2020), we identify a fresh perspective on how this dependence can further be problematic. Namely, we hypothesize that the issue arises due to the NLL loss scaling down the gradient of poorly-predicted data points relative to the well-predicted ones, leading to effectively undersampling the poorly-predicted data points.

We then introduce an *alternative loss formulation* that counteracts this by weighting the contribution of each data point to the overall loss by its β-exponentiated variance estimate, *where β controls the extent of dependency of gradients on predictive variance*. This formulation subsumes the standard NLL loss for $\beta = 0$ and allows to lessen the dependency of gradients on the variance estimates for $0 < \beta \leq 1$. Interestingly, using $\beta = 1$ completely removes such dependency for training the mean estimator, yielding the standard mean squared error (MSE) loss – but with the additional capacity of uncertainty estimation. Finally, we empirically show that our modified loss formulation largely mitigates the issue of poor fits, achieving considerable improvements on a set of standard benchmarks while exhibiting more robustness to network architecture and learning rate configurations.

2 Preliminaries

Let X, Y be two random variables describing the input and target, following the joint distribution $P(X, Y)$. We assume that Y is conditionally independent given X and that it follows some probability distribution $P(Y \mid X)$. In the following, we use the common assumption that Y is normally distributed given X; i.e. $P(Y \mid X) = \mathcal{N}(\mu(X), \sigma^2(X))$, where $\mu\colon \mathbb{R}^M \mapsto \mathbb{R}$ and $\sigma^2\colon \mathbb{R}^M \mapsto \mathbb{R}^+$ are respectively the true input-dependent mean and variance functions.[1] Equivalently, we can write $Y = \mu(X) + \epsilon(X)$, with $\epsilon(X) \sim \mathcal{N}(0, \sigma^2(X))$; i.e. Y is generated from X by $\mu(X)$ plus a zero-mean Gaussian noise with variance $\sigma^2(X)$. This input-dependent variance quantifies the heteroscedastic uncertainty, or input-dependent aleatoric uncertainty.

[1]For notational convenience, we focus on univariate regression but point out that the work extends to the multivariate case as well.

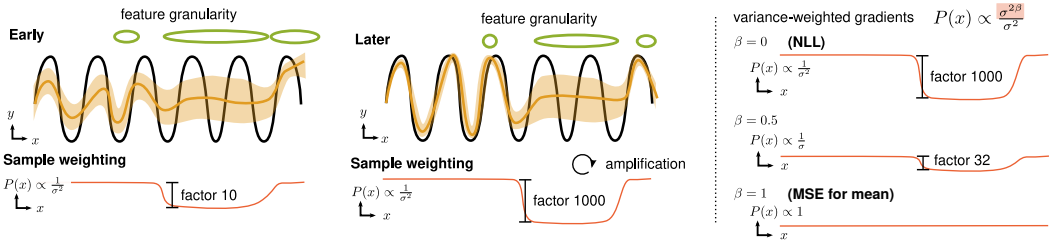

Figure 2: Illustration of the pitfall when training with NLL (negative log-likelihood) and our solution. An initial inhomogeneous feature space granularity (explained in the text) results early on in different fitting quality. The implicit weighting of the squared error in NLL can be seen as a biased data-sampling with $p(x) \sim \frac{1}{\sigma^2(x)}$ (see Eq. 6). Badly fit parts are increasingly ignored during training. On the right, the effect of our solution (Eq. 9) on the relative importance of data points is shown.

To learn estimates $\hat{\mu}(X), \hat{\sigma}^2(X)$ of the true mean and variance functions, it is common to use a neural network f_θ parameterized by θ. Here, $\hat{\mu}(X)$ and $\hat{\sigma}^2(X)$ can be outputs of the final layer (Nix & Weigend, 1994) or use two completely separate networks (Detlefsen et al., 2019).The variance output is hereby constrained to the positive region using a suitable activation function, e.g. softplus. The optimal parameters θ^*_{NLL} can then be found using maximum likelihood estimation (MLE) by minimizing the negative log-likelihood (NLL) criterion $\mathcal{L}_{\text{NLL}}$ under the distribution $P(X, Y)$:

$$\theta^*_{\text{NLL}} = \arg\min_\theta \mathcal{L}_{\text{NLL}}(\theta; \mathcal{D}) = \arg\min_\theta \mathop{\mathbb{E}}_{X,Y}\left[\frac{1}{2}\log\hat{\sigma}^2(X) + \frac{(Y-\hat{\mu}(X))^2}{2\hat{\sigma}^2(X)} + \text{const}\right]. \tag{1}$$

In contrast, standard regression minimizes the mean squared error (MSE) $\mathcal{L}_{\text{MSE}}$:

$$\theta^*_{\text{MSE}} = \arg\min_\theta \mathcal{L}_{\text{MSE}}(\theta; \mathcal{D}) = \arg\min_\theta \mathop{\mathbb{E}}_{X,Y}\left[\frac{(Y-\hat{\mu}(X))^2}{2}\right]. \tag{2}$$

In practice, Eq. 1 and Eq. 2 are optimized using stochastic gradient descent with batches of samples drawn from $P(X, Y)$. The gradients of $\mathcal{L}_{\text{NLL}}$ with respect to $\hat{\mu}(X)$, $\hat{\sigma}^2(X)$ are given by

$$\nabla_{\hat{\mu}}\mathcal{L}_{\text{NLL}}(\theta) = \mathop{\mathbb{E}}_{X,Y}\left[\frac{\hat{\mu}(X)-Y}{\hat{\sigma}^2(X)}\right], \quad \nabla_{\hat{\sigma}^2}\mathcal{L}_{\text{NLL}}(\theta) = \mathop{\mathbb{E}}_{X,Y}\left[\frac{\hat{\sigma}^2(X)-(Y-\hat{\mu}(X))^2}{2\big(\hat{\sigma}^2(X)\big)^2}\right]. \tag{3, 4}$$

3 Analysis

We now return to the example of trying to fit a sinusoidal function from Sec. 1. Recall from Fig. 1 that using the Gaussian NLL as the objective resulted in a suboptimal fit. In contrast, using MSE as the objective, the model converged without problems in reasonable time. We now analyze the reasons behind this surprising result.

From Eq. 3, we see that the true mean $\mu(X)$ is a minimizer of the NLL loss. It thus becomes clear that a) the solution found in Fig. 1 is not the optimal one, and b) the NLL objective should, in principle, drive $\hat{\mu}(X)$ to the optimal solution $\mu(X)$. So why is the model not converging? We identify two main culprits for the non-convergence of the Gaussian NLL objective:

1. Initial flatness of the feature space can create an undercomplex but locally stable mean fit. This fit results from local symmetries and requires a form of symmetry breaking to escape.
2. The NLL loss scales the gradient of badly-predicted points down relative to well-predicted points, effectively undersampling those points. This effect worsens as training progresses.

These culprits and their effect on training are illustrated in Fig. 2 (left). If the network cannot fit a certain region yet, because its feature space (spanned by the last hidden layer) is too coarse, it results in a high effective data variance. This leads to down-weighting the data from these regions, fueling a vicious circle of self-amplifying the increasingly imbalanced weighting. In the following, we analyse these effects and their reasons in more detail.

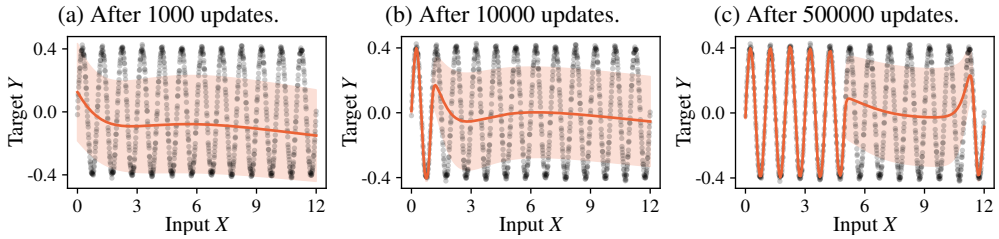

Figure 3: Model fit at different stages of training using the NLL loss function shown in red with $\pm\sigma$ uncertainty band. Black dots mark training data. Fitting the function begins left and is visibly slow.

3.1 Symmetry and Feature Non-Linearity

It is instructive to see how the model evolves over training as shown in Fig. 3. The network first learns essentially the best linear fit while adapting the variance to match the residuals. The situation is locally stable. That is, due to the symmetries of errors below and above the mean fit, there is no incentive to change the situation. Symmetry breaking is required for further progress. One form of symmetry breaking comes from the stochasticity of mini-batch sampling in SGD, or natural asymmetries contained in the dataset, e.g. outlier points. In addition, we hypothesize that the local non-linearity of the feature space plays an important role in creating the necessary non-linear fit.

Thus, let us consider the non-linearity of the feature space. This quantity is not easy to capture. To approximate it, we compute how much the Jacobian J_f of the features $f(x)$ with respect to the input varies in an L2-ball with radius r around a point x, denoted as the Jacobian variance:[2]

$$V(x) = \frac{1}{|\mathcal{B}|} \sum_{x' \in \mathcal{S}} \left(J_f(x') - \frac{1}{|\mathcal{S}|} \sum_{x'' \in \mathcal{B}} J_f(x'') \right)^2, \quad \mathcal{B} = \{x' \colon \|x - x'\|_2 \leq r\}. \tag{5}$$

Figure 4 visualizes the Jacobian variance over the input space as a function of the training progress. Although initially relatively flat, it becomes more fine-granular in parts of the input space – the parts which are later well fit. The region with low Jacobian variance remains stuck in this configuration, see Fig. 1. This provides evidence that the non-linearity of the feature space is important for the success or failure of learning. However, why does gradient descent not break out of this situation?

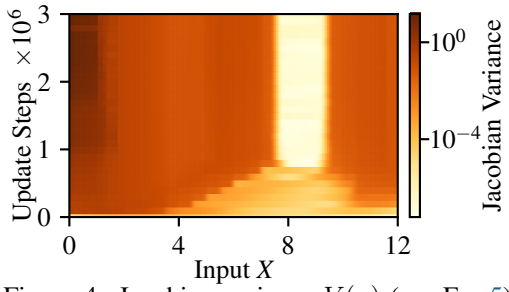

Figure 4: Jacobian variance $V(x)$ (see Eq. 5) over training time.

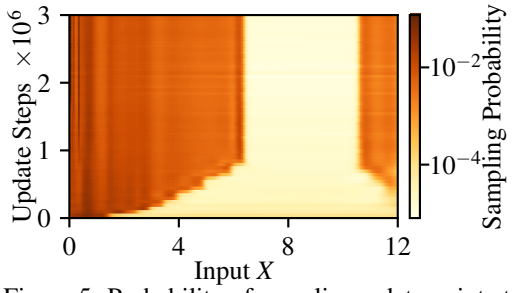

Figure 5: Probability of sampling a data point at x over training time.

3.2 Weighting by Variance Effectively Undersamples

The answer lies in an imbalanced weighting of data points across the input space. Recall that the gradient $\nabla_\mu \mathcal{L}_{\text{NLL}}$ of the NLL with respect to the mean scales the error $\hat{\mu}(X) - Y$ by $\frac{1}{\hat{\sigma}^2(X)}$ (Eq. 3). As symmetry is broken and the true function starts to be fit locally, the variance quickly shrinks in these areas to match the reduced MSE. If the variance is well-calibrated, the gradient becomes $\frac{\hat{\mu}(X)-Y}{\hat{\sigma}^2(X)} \approx \frac{\hat{\mu}(X)-Y}{(\hat{\mu}(X)-Y)^2} = \frac{1}{\hat{\mu}(X)-Y}$. Data points with already low error will get their contribution in the batch gradient scaled up relatively to high error data points – a "rich get richer" self-amplification.

[2]This is a form of second-order derivative computed numerically, which also gives non-zero results for networks with ReLU activation (in contrast to, for example, the Hessian).

Thus, NLL acts opposite to MSE which focuses on high-error samples. If the true variance σ^2 on the well-fit regions is much smaller than the errors on the badly-fit regions, or there are much more well-fit than badly-fit points, then *learning progress is completely prevented* on the badly-fit regions.

Another way to view this is to interpret the different weighting of points as *changing the training distribution* $P(X, Y)$ *to a modified distribution* $\tilde{P}(X, Y)$ in which points with high error have a lower probability of getting sampled. This can be shown by defining $\tilde{P}(X, Y) = Z^{-1} \frac{P(X,Y)}{\sigma^2(X)}$, where $Z = \int \frac{P(x,y)}{\sigma^2(x)} \mathrm{d}x\mathrm{d}y$ is a normalizing constant, and recognizing that the gradient of the NLL is proportional to the gradient of the MSE loss in Eq. 2 under the modified data distribution $\tilde{P}(X, Y)$:

$$\nabla_\mu \mathcal{L}_{\text{NLL}}(\theta) = Z \cdot \mathbb{E}_{(X,Y)\sim\tilde{P}(X,Y)}[\mu(X) - Y] \propto \nabla_\mu \mathbb{E}_{(X,Y)\sim\tilde{P}(X,Y)} \left[\frac{(Y - \mu(X))^2}{2} \right]. \quad (6)$$

In Fig. 5, we plot $\tilde{P}(X, Y)$ over training time for our sinusoidal example. It can be seen that the virtual probability of sampling a point from the high-error region drops over time, until it is highly unlikely to sample points from this region (10^{-5} as opposed to 10^{-3} for uniform sampling). We show that this behavior also carries over to a real-world dataset in Sec. A.2. As a side note, optimization with SGD typically assumes a fixed training distribution; training under shifting distributions may thus make optimization with SGD more difficult. It also was recently shown that such a shifting training setup can hurt generalization (Igl et al., 2021). These two points may further contribute to the suboptimal performance of probabilistic neural networks experienced in practice.

Sometimes "weighting-by-variance" is seen as a feature of the Gaussian NLL (Kendall & Gal, 2017) that introduces a self-regularizing property by allowing the network to "ignore" outlier points with high error. This can be desirable if the predicted variance corresponds to the inherent unpredictability (noise), but it is undesirable if it causes premature convergence and ignorance of hard-to-fit regions, as shown above. In our method (Sec. 4), we allow to control the extent of self-regularization.

4 Method

To mitigate these problems with NLL training, we develop a solution in the following. We start with a method that uses standard squared error losses to estimate the moments of the target distribution, which we term "moment matching" (MM; Sec. 4.1). This method fixes the problem with premature convergence when using $\mathcal{L}_{\text{NLL}}$ and yields consistent training results, but it also leads to underestimated variances and will act as one of the baselines in our empirical evaluations. In Sec. 4.2, we then consider the distribution of residual errors which are drastically different between NLL training and MSE training. This suggests a problem-specific design choice that leads us to our proposed solution, which we term $\beta-$NLL, in Sec. 4.3. This solution method allows choosing a meaningful loss-interpolation between NLL and MSE while keeping calibrated uncertainty estimates.

4.1 Moment Matching

A natural solution to the problem of estimating distributions would be to estimate the sufficient statistics of the distribution. In the case of the Gaussian distribution, these would be the first two moments μ, σ^2 fully describing the distribution. So why not defining losses based on the moment estimators? Starting from the conditional mean of the predictions y: $\mathbb{E}[Y \mid X]$, we can define the squared deviation as a loss and see that the MSE is an upper bound:

$$(\mathbb{E}[Y \mid X] - \hat{\mu}(X))^2 = \mathbb{E}[(Y - \hat{\mu}(X)) \mid X]^2 \leq \mathbb{E}\big[(Y - \hat{\mu}(X))^2 \mid X\big] := \mathcal{L}^{\mu}_{\text{MM}}. \quad (7)$$

Notice that $\mathcal{L}^{\mu}_{\text{MM}}$ is the standard MSE loss. Thus, fitting the mean $\hat{\mu}(x)$ can follow standard (non-distributional) regression procedures. Interestingly, due to the moment-matching viewpoint, we can analogously define a loss for the variance (second central moment). The variance is defined as $\mathbb{E}\big[(Y - \mu(X))^2 \mid X\big]$; analogously, we get the following loss: $\mathbb{E}\Big[\big((Y - \hat{\mu}(X))^2 - \hat{\sigma}^2(X)\big)^2 \mid X\Big]$. To use the same physical unit as $\mathcal{L}^{\mu}_{\text{MM}}$, we reformulate it in terms of the standard deviation as:

$$\mathcal{L}^{\sigma}_{\text{MM}} := \mathbb{E}\left[\left(\sqrt{(Y - \hat{\mu}(X))^2} - \hat{\sigma}(X) \right)^2 \,\middle|\, X \right]. \quad (8)$$

Thus, the moment matching loss is simply given by the sum of both losses: $\mathcal{L}_{\text{MM}} = \mathcal{L}^{\mu}_{\text{MM}} + \mathcal{L}^{\sigma}_{\text{MM}}$.

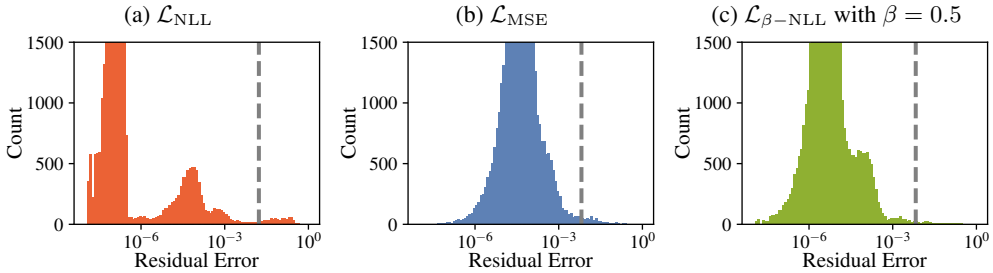

Figure 6: Distribution over residual prediction errors depending on the choice of loss function for the ObjectSlide dataset (see Sec. 5.2). Dashed line shows the RMSE. (a) NLL loss: residuals are multimodal. There is a long tail of difficult data points that are ignored, whereas easy regions are fit to high accuracy. (b) Using MSE loss results in a log-normal residual distribution. (c) Our loss $\mathcal{L}_{\beta-\text{NLL}}$, introduced below, yields highly accurate fits in easy regions without ignoring hard examples.

4.2 Residual Distribution and Allocation of Function Approximator Capacity

The moment matching loss gives the same weighting to all data points. Since we are dealing with function approximators, the question is whether their capacity should be used equally. As discussed in Sec. 3.2, the NLL loss gives high weight to data points with low (predicted) variance and low weight to those with high variance. This is appropriate in case these variances are caused by true aleatoric uncertainty in the data. However, due to the use of function approximators, there is also the case where data points cannot be well predicted, resulting in high predicted variance, although the ground truth is corrupted by little noise. The handling of these *difficult points* is different for the different loss functions. Figure 6 shows the distribution of the residuals for a dynamics prediction dataset containing *easy* and *hard* to model areas. We observe a drastic difference between the NLL loss and the MSE loss (or equivalently solely optimizing $\mathcal{L}^{\mu}_{\text{MM}}$).

How important are the data points with high uncertainty? Are these outliers or stem from truly noisy regions? In this case, we would be willing to allocate less of the function approximator's capacity to them. Or are these simply difficult samples? In our dynamics prediction example, we analysed the data points with high uncertainty and found that they were actually *the most important ones* to get correct (because they captured non-trivial interactions in the physical world). Following this insight, this suggests that there is no one-loss-fits-all, but rather that the modeler should have the choice to select which behavior is desired for the application at hand.

4.3 Variance-weighting the gradients of the NLL

In this section, we introduce our proposed solution to the problems occurring when training with $\mathcal{L}_{\text{NLL}}$. The most important problem to solve is the premature convergence of NLL training to highly suboptimal mean fits. In Sec. 3, we identified the relative down-weighting of badly fit data points with its self-amplifying character as the main reason for this. Effectively, NLL weights the mean squared error per data point with $\frac{1}{\sigma^2}$, which can be interpreted as sampling data points with $P(x) \propto \frac{1}{\sigma^2}$ (Sec. 3.2). Consequently, we propose modifying this distribution by introducing a parameter β allowing to *interpolate* between NLL's and a completely uniform data point importance. The resulting sampling distribution is given by $P(x) \propto \frac{\sigma^{2\beta}}{\sigma^2}$ and illustrated in Fig. 2 (right).

How could this weighting be achieved? We simply introduce the variance-weighting term $\sigma^{2\beta}$ to the $\mathcal{L}_{\text{NLL}}$ loss such that it acts as a factor on the gradient. We denote the resulting loss as $\beta-$NLL:

$$\mathcal{L}_{\beta-\text{NLL}} := \mathop{\mathbb{E}}_{X,Y}\left[\lfloor\hat{\sigma}^{2\beta}(X)\rfloor\left(\frac{1}{2}\log\hat{\sigma}^2(X) + \frac{(Y-\hat{\mu}(X))^2}{2\hat{\sigma}^2(X)} + \text{const}\right)\right], \tag{9}$$

where $\lfloor\cdot\rfloor$ denotes the *stop gradient* operation. By stopping the gradient, the variance weighting term acts as an adaptive, input-dependent learning rate. In this way, the gradients of $\mathcal{L}_{\beta-\text{NLL}}$ are:

$$\nabla_{\hat{\mu}}\mathcal{L}_{\beta-\text{NLL}}(\theta) = \mathop{\mathbb{E}}_{X,Y}\left[\frac{\hat{\mu}(X)-Y}{\hat{\sigma}^{2-2\beta}(X)}\right], \nabla_{\hat{\sigma}^2}\mathcal{L}_{\beta-\text{NLL}}(\theta) = \mathop{\mathbb{E}}_{X,Y}\left[\frac{\hat{\sigma}^2(X)-(Y-\hat{\mu}(X))^2}{2\hat{\sigma}^{4-2\beta}(X)}\right]. \tag{10, 11}$$

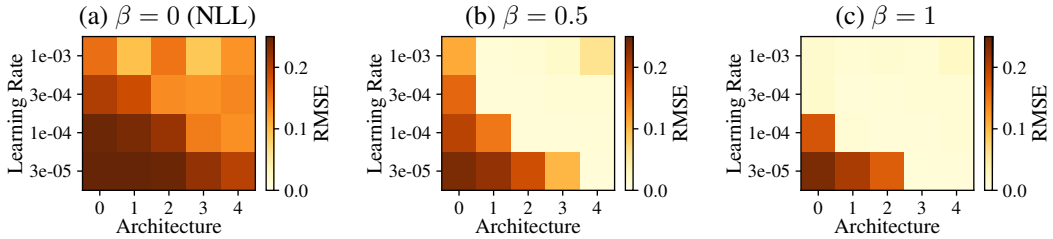

Figure 7: Convergence properties analyzed on the sinusoidal toy regression problem. The root mean squared error (RMSE) is displayed as a color code, depending on model-architecture (see Sec. C.1) and learning rate after 200 000 epochs (mean over 3 independent trials). The original NLL ($\beta = 0$) does not obtain good RMSE fits for most hyperparameter settings. Figure S3 shows the NLL scores.

Naturally, for $\beta = 0$, we recover the original NLL loss. For $\beta = 1$ the gradient w.r.t. μ in Eq. 10 is equivalent to the one of MSE. However, for the variance, the gradient in Eq. 11 is a new quantity with $2\sigma^2$ in the denominator. This is in contrast to the moment matching loss (Eq. 8). For values $0 < \beta < 1$, we get different loss interpolations. Particularly interesting is the case of $\beta = 0.5$, where the data points are weighted with $\frac{1}{\sigma}$ (inverse standard deviation instead of inverse variance). In Sec. 5, we show that $\beta = 0.5$ generally achieves the best trade-off between accuracy and log-likelihood. A Pytorch implementation of the loss function is provided in Sec. C.5.

As a remark, the new loss $\mathcal{L}_{\beta-\mathrm{NLL}}$ is not meant for performance evaluation – it is designed to result in meaningful gradients. We found that its absolute value can be counter-intuitive due to the prefactor. The model performance during training should be monitored with the original (negative) log-likelihood and optionally the RMSE for the mean fit.

5 Experiments

We investigate the proposed changes in the loss function for training neural networks to predict a Gaussian distribution on synthetic examples and real-world datasets featuring the UCI, challenging dynamics model datasets, and regression tasks on MNIST, Fashion-MNIST, and depth-map prediction.

As motivated above, the standard NLL loss often leads to suboptimal fits, e.g. where the mean fit is far from ideal. To analyze this quantitatively, we report the performance for many hyperparameter settings, such as learning rates and network architectures, and consider both the quality in predictions with respect to the root mean squared error (RMSE) and the negative log-likelihood (NLL).

5.1 Synthetic Datasets

Sinusoidal without heteroscedastic noise We first consider our illustrative example of Fig. 1 – a simple sine curve with a small additive noise: $y = \sin(x) + \xi$, with ξ being Gaussian noise with standard deviation $\sigma = 0.01$. One would expect that a network with sufficient capacity can easily learn to fit this function. Figure 7 inspects this over a range of architectures and learning rates.

We find that for the standard NLL ($\beta = 0$) the network does not converge to a reasonable mean fit. There is a trend that larger networks and learning rates show better results, but when comparing this to $\beta-$NLL with $\beta = 0.5$ we see that the networks are indeed able to fit the function without problems. As expected, the same holds for the mean squared error loss (MSE) and $\beta-$NLL with $\beta = 1$. The quality of the fit w.r.t. NLL is shown in Fig. S3 for completeness.

Sinusoidal with heteroscedastic noise A commonly used illustrative example introduced in Detlefsen et al. (2019) is a sine curve with increasing amplitude and noise: $y = x\sin(x) + x\xi_1 + \xi_2$, with ξ_1 and ξ_2 being Gaussian noise with standard deviation $\sigma = 0.3$. The functional form is much easier, so fitting the mean is achieved with all losses. Here we sanity-check that the newly introduced loss is still delivering good uncertainty estimates. Figure 8 (a-e) displays the predictions of the best models (w.r.t. NLL validation loss) and (f) compares the predicted uncertainties over 10 independent trials.

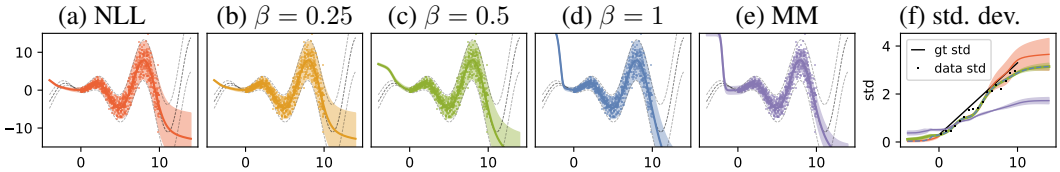

Figure 8: Fits for the heteroscedastic sine example from Detlefsen et al. (2019) (a-e). Dotted lines show the ground truth mean and $\pm 2\sigma$, respectively. (f) The predicted standard deviations (with shaded std. over 10 independent trials) with the same color code. Note that $\beta = 0.5$ and $\beta = 1$ are lying on top of another. Inside the training regime, all $\beta-$NLL variants (a-d) yield well-calibrated uncertainty estimates. Moment matching (e) is significantly underestimating the variance everywhere.

Table 1: Results for UCI Regression Datasets. We report predictive log-likelihood and RMSE ($\pm$ standard deviation). *Ties* gives the number of datasets (of 12) for which the method cannot be statistically distinguished from the best method (see Sec. A.3). We compare with Student-t (Detlefsen et al., 2019) and xVAMP/VBEM (Stirn & Knowles, 2020). Sec. A.3 lists the full results.

		LL ↑					RMSE ↓				
Loss	β	Ties	concrete	energy	naval	yacht	Ties	concrete	energy	naval	yacht
$\mathcal{L}_{\beta-\text{NLL}}$	0	2	-3.25 ± 0.31	-2.44 ± 0.36	12.74 ± 0.47	-2.86 ± 5.18	4	6.08 ± 0.65	2.21 ± 0.21	0.0022 ± 0.0006	1.22 ± 0.47
$\mathcal{L}_{\beta-\text{NLL}}$	0.25	3	-3.31 ± 0.51	-2.13 ± 0.22	13.63 ± 0.36	-1.97 ± 1.14	5	5.79 ± 0.74	1.82 ± 0.18	0.0012 ± 0.0004	1.73 ± 1.00
$\mathcal{L}_{\beta-\text{NLL}}$	0.5	5	-3.29 ± 0.36	-1.89 ± 0.20	13.77 ± 0.58	-2.47 ± 1.68	6	5.61 ± 0.65	1.10 ± 0.10	0.0006 ± 0.0002	2.35 ± 1.44
$\mathcal{L}_{\beta-\text{NLL}}$	0.75	5	-3.27 ± 0.34	-2.05 ± 0.27	13.74 ± 0.44	-1.87 ± 0.55	7	5.67 ± 0.73	1.04 ± 0.12	0.0004 ± 0.0001	1.97 ± 1.03
$\mathcal{L}_{\beta-\text{NLL}}$	1	2	-3.23 ± 0.33	-2.89 ± 0.61	13.59 ± 0.38	-2.27 ± 1.07	8	5.55 ± 0.77	1.44 ± 0.38	0.0004 ± 0.0001	2.08 ± 1.13
$\mathcal{L}_{\text{MM}}$		0	-3.49 ± 0.38	-4.05 ± 0.25	-2.46 ± 66.93	-11.2 ± 31.0	4	6.28 ± 0.82	2.38 ± 0.23	0.0005 ± 0.0001	3.02 ± 1.38
$\mathcal{L}_{\text{MSE}}$		—	—	—	—	—	11	4.96 ± 0.64	0.95 ± 0.08	0.0003 ± 0.0001	0.78 ± 0.25
Student-t		10	-3.07 ± 0.14	-2.22 ± 0.25	12.49 ± 0.50	-1.23 ± 0.55	5	5.82 ± 0.59	2.22 ± 0.18	0.0023 ± 0.0007	1.34 ± 0.63
xVAMP		7	-3.06 ± 0.15	-2.34 ± 0.15	12.66 ± 0.55	-0.99 ± 0.33	7	5.44 ± 0.64	2.06 ± 0.19	0.0019 ± 0.0008	0.99 ± 0.43
xVAMP*		7	-3.03 ± 0.13	-2.30 ± 0.22	12.85 ± 0.59	-1.04 ± 0.47	8	5.35 ± 0.73	2.03 ± 0.13	0.0017 ± 0.0006	1.13 ± 0.66
VBEM		2	-3.14 ± 0.07	-4.33 ± 0.18	8.14 ± 0.12	-2.65 ± 0.10	8	5.21 ± 0.58	1.36 ± 0.36	0.0010 ± 0.0005	1.66 ± 0.84
VBEM*		8	-2.99 ± 0.13	-1.92 ± 0.24	12.81 ± 0.98	-0.98 ± 0.24	8	5.17 ± 0.59	1.26 ± 0.23	0.0018 ± 0.0006	0.65 ± 0.20

5.2 Real-world Datasets

UCI Regression Datasets As a standard real-world benchmark in predictive uncertainty estimation, we consider the UCI datasets (Hernández-Lobato & Adams, 2015). Table 1 gives an overview comparing different loss variants. We refer to Sec. A.3 for the full results on all 12 datasets. The results are encouraging: $\beta-$NLL achieves predictive log-likelihood on par with or better than the NLL loss while clearly improving the predictive accuracy on most datasets.

Dynamics models As a major application of uncertainty estimation lies in model-based reinforcement learning (RL), we test the different loss functions on two dynamics predictions tasks of varying difficulty, ObjectSlide and Fetch-PickAndPlace. In both tasks, the goal is to predict how an object will move from the current state and the agent's action. Whereas ObjectSlide (Seitzer et al., 2021) is a simple 1D-environment, Fetch-PickAndPlace (Plappert et al., 2018) is a complex 3D robotic-manipulation environment. The models are trained on state-action trajectories collected by RL agents. See Sec. B for more details.

For both datasets and all loss functions, we perform a grid search over different hyperparameter configurations (see Sec. C.2) for a sensitivity analysis to hyperparameters settings, presented in Fig. 9. It reveals that both the NLL- and the MM-loss are vulnerable to wrong hyperparameter settings, whereas the $\beta-$NLL loss achieves good results over a wide range of settings. The best performing configurations for each loss are evaluated on a hold-out test set (Table 2). One can see that the NLL loss results in poor predictive performance and also exhibits quite a high variance across random seeds; the MM-loss results in poor log-likelihood fits. Our method yields high accuracy and log-likelihood fits, for a range of β-values, where $\beta = 0.5$ generally achieves the best trade-off.

Generative models and depths-map prediction In Fig. S5 we present results on variational autoencoders with probabilistic decoders, showing qualitatively similar trends as on UCI. More details are provided in Sec. A.4. On the task of depth regression we modify a state-of-the-art method (Bhat et al., 2021) and test it on the NYUv2 dataset (Silberman et al., 2012) with our loss (Fig. S6). β-NLL yields an improved mean fit in comparison to plain NLL while also producing sharper depth maps, as detailed in Sec. A.5.

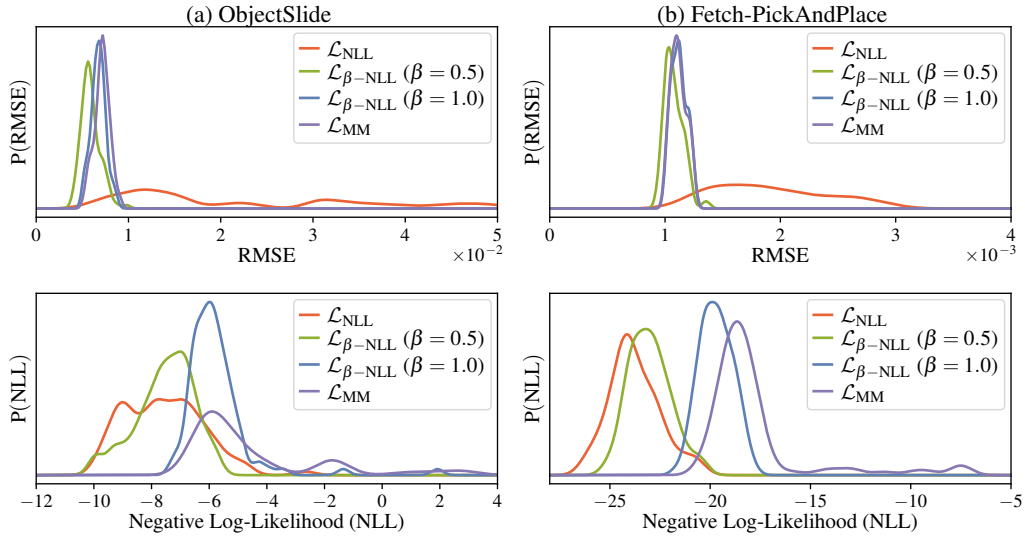

Figure 9: Sensitivity analysis of loss functions to hyperparameter settings. Distribution over RMSE (top row) and NLL (bottom row) as a function of hyperparameters, on validation sets of the ObjectSlide (a) and Fetch-PickAndPlace (b) task. The values stem from a grid search over different model configurations (see C.2). The NLL loss is sensitive when evaluating RMSE, whereas the MM loss is sensitive when evaluating NLL (many runs for MM ended with exploding NLLs). In contrast, the β-NLL loss shows much less sensitivity and achieves stable results regardless of the concrete settings.

Table 2: Test results for dynamics models, using best configurations found in grid search. Reported standard deviation is over 5 random seeds. We compare with Student-t (Detlefsen et al., 2019) and xVAMP/VBEM (Stirn & Knowles, 2020).

		1D-Slide		**Fetch-PickAndPlace**	
Loss	β	RMSE ↓	LL ↑	RMSE ↓	LL ↑
$\mathcal{L}_{\beta-\text{NLL}}$	0	0.0192 ± 0.006	7.97 ± 3.62	0.00163 ± 0.00008	18.72 ± 7.32
$\mathcal{L}_{\beta-\text{NLL}}$	0.25	0.0107 ± 0.004	9.03 ± 0.47	0.00102 ± 0.00004	24.43 ± 1.64
$\mathcal{L}_{\beta-\text{NLL}}$	0.5	0.0064 ± 0.002	9.28 ± 0.75	0.00096 ± 0.00002	24.68 ± 0.08
$\mathcal{L}_{\beta-\text{NLL}}$	0.75	0.0087 ± 0.003	6.61 ± 1.83	0.00098 ± 0.00001	22.77 ± 0.17
$\mathcal{L}_{\beta-\text{NLL}}$	1.0	0.0074 ± 0.001	6.58 ± 0.29	0.00102 ± 0.00001	21.32 ± 0.07
$\mathcal{L}_{\text{MM}}$		0.0078 ± 0.001	diverges	0.00104 ± 0.00003	19.33 ± 1.31
$\mathcal{L}_{\text{MSE}}$		0.0068 ± 0.001	—	0.00103 ± 0.00000	—
Student-t		0.0155 ± 0.006	11.30 ± 0.03	0.00117 ± 0.00001	30.44 ± 0.08
xVAMP		0.0118 ± 0.002	10.58 ± 0.19	0.00128 ± 0.00005	29.02 ± 0.12
xVAMP*		0.0199 ± 0.006	10.89 ± 0.10	0.00128 ± 0.00001	29.19 ± 0.08
VBEM		0.0039 ± 0.000	3.79 ± 0.00	0.00104 ± 0.00003	17.39 ± 0.29
VBEM*		0.0280 ± 0.011	10.13 ± 0.49	0.00118 ± 0.00003	28.62 ± 0.15

6 Discussion

We highlight a deep problem frequently occurring when optimizing probabilistic neural networks using the common NLL loss: training gets stuck in suboptimal function fits. Due to the predictive variance-based weighting of mean prediction errors in the NLL loss, these bad fits often stay unnoticed, unless the MSE is evaluated. With our analysis on feature space non-linearity and induced data-set weighting (interpreted as a biased dataset sampling), we reveal the underlying reason: initially badly fit regions (due to low feature non-linearity) get increasingly less weight and result in premature convergence. We propose a simple solution by introducing a family of loss functions called $\beta-$NLL. Effectively, the gradient of the original NLL loss is scaled by the β-exponentiated per-sample variance. This allows for a meaningful interpolation between NLL loss and MSE loss while providing well-behaved uncertainty estimates (even for the MSE case). The hyperparameter β gives practitioners the choice to control the self-regularization strength of NLL: how unimportant should high-noise regions or difficult-to-predict data points be in the fitting process. In most cases, $\beta = 0.5$ will be a good starting point. We hope that our simple solution will improve the usability and performance of predictive uncertainty deep networks and increase their applicability.

REPRODUCIBILITY STATEMENT

All settings are described in detail in Sec. B and Sec. C. We make full code and data available under https://sites.google.com/view/pitfalls-uncertainty. Both will be made publicly available after the review phase.

REFERENCES

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *ArXiv*, abs/1606.06565, 2016.

Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. Adabins: Depth estimation using adaptive bins. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4009–4018, 2021.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1613–1622, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/blundell15.html.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, J. Schneider, John Schulman, Jie Tang, and W. Zaremba. OpenAI Gym. *ArXiv*, abs/1606.01540, 2016.

Xiaotian Chen, X. Chen, and Zhengjun Zha. Structure-aware residual pyramid network for monocular depth estimation. In *IJCAI*, 2019.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/3de568f8597b94bda53149c7d7f5958c-Paper.pdf.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1096–1105. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/dabney18a.html.

Nicki S. Detlefsen, Martin Jørgensen, and Søren Hauberg. Reliable training and estimation of variance networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/07211688a0869d995947a8fb11b215d6-Paper.pdf.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/gal16.html.

Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep Bayesian active learning with image data. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1183–1192. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/gal17a.html.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1321–1330. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/guo17a.html.

Ali Harakeh and Steven L. Waslander. Estimating and evaluating regression predictive uncertainty in deep object detectors. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YLewtnvKgR7.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Hkg4TI9xl.

José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1861–1869. JMLR.org, 2015.

Shi Hu, Nicola Pezzotti, and Max Welling. A new perspective on uncertainty quantification of deep ensembles. *ArXiv*, 2020.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Qun8fv4qSby.

Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/2650d6089a6d640c5e85b2b88265dc2b-Paper.pdf.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009. ISSN 0167-4730. doi: https://doi.org/10.1016/j.strusafe.2008.06.020. URL https://www.sciencedirect.com/science/article/pii/S0167473008000556. Risk Acceptance and Risk Communication.

Alina Kloss, Georg Martius, and Jeannette Bohg. How to train your differentiable filter. *Autonomous Robots*, 45:562–578, June 2021. doi: 10.1007/s10514-021-09990-9. URL https://link.springer.com/article/10.10072Fs10514-021-09990-9.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf.

Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. *ArXiv*, abs/1907.10326, 2019.

Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip H. S. Torr, and Yarin Gal. Deterministic neural networks with appropriate inductive biases capture epistemic and aleatoric uncertainty. *CoRR*, abs/2102.11582, 2021. URL https://arxiv.org/abs/2102.11582.

David A. Nix and Andreas S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pp. 55–60 vol.1, 1994. doi: 10.1109/ICNN.1994.374138.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/8d8818c8e140c64c743113f563cf750f-Paper.pdf.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/8558cb408c1d76621371888657d2eb1d-Paper.pdf.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, and Georg Martius. Extracting strong policies for robotics tasks from zero-order trajectory optimizers. In *9th International Conference on Learning Representations (ICLR 2021)*, May 2021.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, J. Schneider, Joshua Tobin, Maciek Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *ArXiv*, abs/1802.09464, 2018.

Maximilian Seitzer, Bernhard Schölkopf, and Georg Martius. Causal influence detection for improving efficiency in reinforcement learning. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*. Curran Associates, Inc., December 2021. URL https://arxiv.org/abs/2106.03443.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

Andrew Stirn and David A. Knowles. Variational variance: Simple and reliable predictive variance parameterization. *ArXiv*, abs/2006.04910, 2020.

Hiroshi Takahashi, Tomoharu Iwata, Yuki Yamanaka, Masanori Yamada, and Satoshi Yagi. Student-t variational autoencoder for robust density estimation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 2696–2702. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/374. URL https://doi.org/10.24963/ijcai.2018/374.

Marin Vlastelica, Sebastian Blaes, Cristina Pinneri, and Georg Martius. Risk-averse zero-order trajectory optimization. In *Conference on Robot Learning*, 2021.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based offline policy optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 14129–14142. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/a322852ce0df73e204b7e67cbbef0d0a-Paper.pdf.

APPENDIX

A ADDITIONAL RESULTS

A.1 SYNTHETIC DATASET

In Fig. S1, we replicate the experiment from Fig. 1 (training the NLL loss on a sinusoidal for 10^7 updates) on several more random seeds. In order to test the dependence of our reported issue on the optimizer, we repeat the experiment from Fig. 1, but with different optimizers instead of Adam with $\beta_1 = 0.9, \beta_2 = 0.999$. The results are shown in Fig. S2. We find that none of the configurations reaches below an RMSE of 0.1 (the optimal value corresponds to an RMSE of 0.01), indicating that the issue is occurring stably across optimization settings. In Fig. S3, we provide the results for the NLL metric on the sinusoidal dataset, complimentary to the results in Fig. 7 (see discussion in Sec. 5.1).

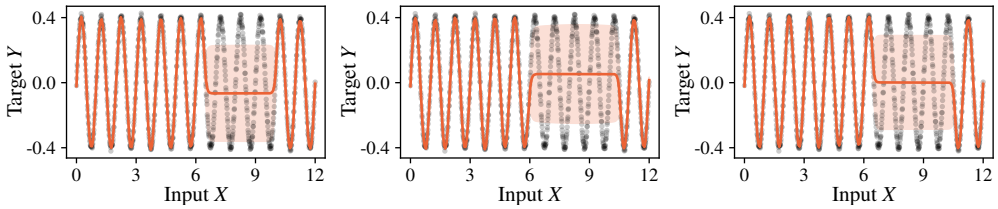

Figure S1: Repeating the experiment from Fig. 1, i.e. training with NLL loss for 10^7 update steps. The observed behavior is stable across different independent trials.

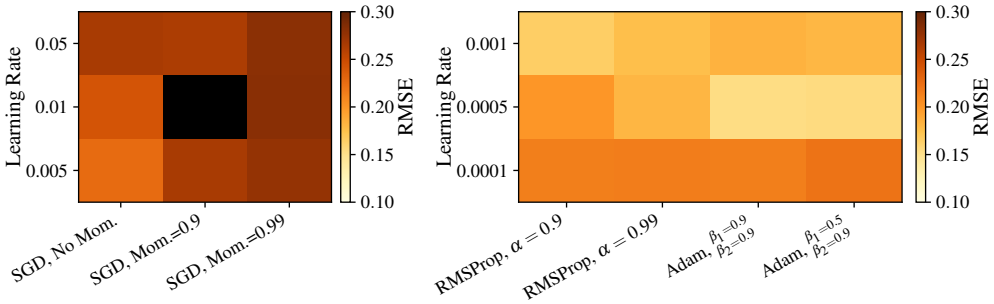

Figure S2: Using different optimizers to train on the sinusoidal from Fig. 1 with NLL loss. The color code indicates the mean RMSE over 3 independent trials per optimizer setting. Black indicates that all trials diverged. Training was done for $2 \cdot 10^6$ update steps and used architecture 2 from Table S5. The observed behavior is stable across optimization settings.

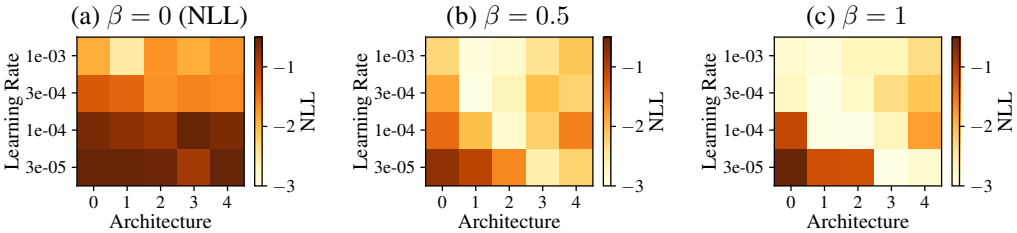

Figure S3: Convergence properties analyzed on the sinusoidal toy regression problem. Same as Fig. 7, but for the negative log-likelihood (NLL). Due to the bad mean fit, the original NLL loss ($\beta = 0$) is also bad for most hyperparameter settings. With $\beta > 0.5$ good fits are obtained for many settings. Also for $\beta = 1$, which corresponds to MSE for fitting the mean, good uncertainty predictions are obtained with our $\beta-$NLL as testified by the low NLL scores.

A.2 Analysis of Sampling Probabilities on Fetch-PickAndPlace

In Fig. S4, we show how the analysis from Sec. 3.2 transfers to a real world dataset, namely Fetch-PickAndPlace. The figure shows how the distribution of effective sampling probability evolves during training (over a fixed set of training points) and compares that against a proxy "oracle": the distribution of effective sampling probabilities when using the squared residuals from a model trained with the MSE loss. T he mismatch between the two distributions demonstrates that optimizing the NLL loss drastically undersamples in comparison to the reference, effectively never sampling some data points. This further corroborates our analysis in Sec. 3.2.

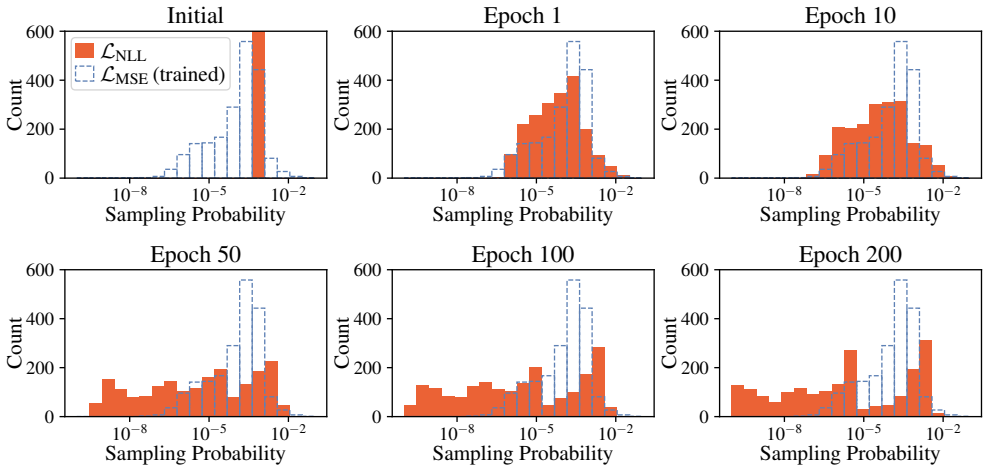

Figure S4: Undersampling behavior of $\mathcal{L}_{\text{NLL}}$ on Fetch-PickAndPlace. The plot shows how the distribution of effective sampling probability evolves over training time, taken over 2 000 fixed training points sampled at the initial epoch. The dashed blue histogram shows the distribution of effective sampling probabilities when using the squared residuals from a model trained with MSE loss $\mathcal{L}_{\text{MSE}}$. This gives a reference distribution that $\mathcal{L}_{\text{NLL}}$ should roughly match, taking into account the relative hardness of prediction on different samples. The $\mathcal{L}_{\text{NLL}}$ drastically undersamples compared to the reference (note the log-scale), effectively never sampling some points.

A.3 UCI Datasets

In this section, we include results for predictive log-likelihood (Table S1) and RMSE (Table S2) for all UCI datasets we evaluated on.

We find that baselines based on the Student-t distribution (xVAMP, VBEM (Stirn & Knowles, 2020)) tend to have better predictive log-likelihood than $\beta-$NLL, although there are also datasets where $\beta-$NLL performs better ("energy", "naval"), or there is no statistically significant improvement ("housing", "kin8m", "wine-red"). We conjecture this is because the Student-t distribution might be better suited to model real-world data than the Gaussian, especially in low-data regimes. For RMSE, $\mathcal{L}_{\text{MSE}}$ unsurprisingly performs best. Our $\beta-$NLL is often on-par with $\mathcal{L}_{\text{MSE}}$ and on-par or better than the baselines, with the exception of "yacht". At the same time, our method is very simple to implement (see Sec. C.5) and computationally lightweight compared to xVAMP and VBEM, which require the costly evaluation of a prior and Monte-Carlo sampling at each training step.

A.4 Variational Autoencoders

We test different loss functions on the task of generative modelling using Variational Autoencoders (VAEs) (Kingma & Welling, 2014). To this end, we parametrize the decoder distribution $p(x \mid z)$ with $\mathcal{N}\big(\mu(z), \sigma^2(z)\big)$, where $\mu(z)$ is the mean and $\sigma^2(z)$ the variance output from a neural network. We train the VAE by maximizing the ELBO $\mathbb{E}_{q(z|x)}[\log p(x \mid z)] - D_{\text{KL}}(q(z \mid x) \mid\mid p(z))$, plugging in different loss functions for $\log p(x \mid z)$. Following (Stirn & Knowles, 2020), we evaluate the log-posterior predictive likelihood $\log \mathbb{E}_{q(z|x)}[p(x \mid z)]$. We approximate the expectation using a finite mixture of 20 Monte-Carlo samples from $q(z \mid x)$. To compute the RMSE, we take the mean of

Table S1: Results for UCI Regression Datasets. Predictive log-likelihood (higher is better) and standard deviation, together with dataset size, input and output dimensions. Best mean value in bold. Results that are not statistically distinguishable from the best result are marked with †.

Loss	carbon (10721, 5, 3)	concrete (1030, 8, 1)	energy (768, 8, 2)	housing (506, 13, 1)
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0)$	13.13 ± 0.25	-3.25 ± 0.31	-2.44 ± 0.36	-2.86 ± 0.50
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.25)$	12.99 ± 0.29	-3.31 ± 0.51	-2.13 ± 0.22	-2.75 ± 0.42
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.5)$	12.36 ± 0.84	-3.29 ± 0.36	**-1.89 ± 0.20**	-2.64 ± 0.36†
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.75)$	10.16 ± 4.97	-3.27 ± 0.34	-2.05 ± 0.27†	-2.72 ± 0.42†
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 1.0)$	-0.47 ± 30.22	-3.23 ± 0.33	-2.89 ± 0.61	-2.85 ± 0.87
$\mathcal{L}_{\mathrm{MM}}$	12.02 ± 0.83	-3.49 ± 0.38	-4.05 ± 0.25	-3.42 ± 1.01
$\mathcal{L}_{\mathrm{MSE}}$	—	—	—	—
Student-t	**15.68 ± 0.30**	-3.07 ± 0.14†	-2.22 ± 0.25	-2.47 ± 0.24†
xVAMP	13.20 ± 0.20	-3.06 ± 0.15†	-2.34 ± 0.15	-2.43 ± 0.21†
xVAMP*	13.14 ± 0.25	-3.03 ± 0.13†	-2.30 ± 0.22	-2.45 ± 0.22†
VBEM	5.81 ± 0.65	-3.14 ± 0.07	-4.33 ± 0.18	-2.56 ± 0.15
VBEM*	13.12 ± 0.37	**-2.99 ± 0.13**	-1.92 ± 0.24†	**-2.42 ± 0.22**
Loss	kin8m (8192, 8, 1)	naval (11934, 16, 2)	power (9568, 4, 1)	protein (45730, 9, 1)
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0)$	1.140 ± 0.039†	12.74 ± 0.47	-2.807 ± 0.057	-2.80 ± 0.05
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.25)$	1.142 ± 0.026†	13.63 ± 0.36†	-2.801 ± 0.053	-2.79 ± 0.05
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.5)$	1.141 ± 0.046†	**13.77 ± 0.58**	-2.805 ± 0.052	-2.78 ± 0.02
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.75)$	1.137 ± 0.041†	13.74 ± 0.44†	-2.806 ± 0.053	-2.79 ± 0.02
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 1.0)$	1.126 ± 0.041	13.59 ± 0.38†	-2.810 ± 0.051	-2.80 ± 0.03
$\mathcal{L}_{\mathrm{MM}}$	0.999 ± 0.063	-2.46 ± 66.93	-2.918 ± 0.092	-2.98 ± 0.06
$\mathcal{L}_{\mathrm{MSE}}$	—	—	—	—
Student-t	**1.155 ± 0.037**	12.49 ± 0.50	**-2.738 ± 0.026**	**-2.56 ± 0.02**
xVAMP	1.147 ± 0.037†	12.66 ± 0.55	-2.788 ± 0.032	-2.74 ± 0.03
xVAMP*	1.147 ± 0.036†	12.85 ± 0.59	-2.785 ± 0.036	-2.71 ± 0.02
VBEM	1.019 ± 0.054	8.14 ± 0.12	-2.819 ± 0.018	-2.85 ± 0.01
VBEM*	1.138 ± 0.036†	12.81 ± 0.98	-2.783 ± 0.031	-2.73 ± 0.01
Loss	superconductivity (21263, 81, 1)	wine-red (1599, 11, 1)	wine-white (4898, 11, 1)	yacht (308, 6, 1)
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0)$	-3.75 ± 0.17	-1.03 ± 0.24†	-1.011 ± 0.016	-2.86 ± 5.18
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.25)$	-3.71 ± 0.15	-0.98 ± 0.12†	-1.010 ± 0.016	-1.97 ± 1.14
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.5)$	-3.65 ± 0.07	-0.99 ± 0.16†	-1.009 ± 0.014	-2.47 ± 1.68
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.75)$	-3.71 ± 0.09	-1.02 ± 0.22†	-1.013 ± 0.013	-1.87 ± 0.55
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 1.0)$	-3.84 ± 0.07	-0.97 ± 0.10†	-1.014 ± 0.021	-2.27 ± 1.07
$\mathcal{L}_{\mathrm{MM}}$	-4.30 ± 0.28	-1.22 ± 0.43	-1.116 ± 0.055	-11.24 ± 31.03
$\mathcal{L}_{\mathrm{MSE}}$	—	—	—	—
Student-t	**-3.40 ± 0.04**	-0.94 ± 0.10†	-1.001 ± 0.010†	-1.23 ± 0.55†
xVAMP	-3.43 ± 0.03†	-0.95 ± 0.06†	-0.995 ± 0.016†	-0.99 ± 0.33†
xVAMP*	-3.42 ± 0.02†	**-0.94 ± 0.06**	**-0.992 ± 0.015**	-1.04 ± 0.47†
VBEM	-3.64 ± 0.15	-0.95 ± 0.07†	-1.004 ± 0.013†	-2.65 ± 0.10
VBEM*	-3.42 ± 0.03†	-0.94 ± 0.07†	-0.999 ± 0.017†	**-0.98 ± 0.24**

Table S2: Results for UCI Regression Datasets. RMSE (lower is better) and standard deviation, together with dataset size, input and output dimensions. Best mean value in bold. Results that are not statistically distinguishable from the best result are marked with †.

Loss	carbon (10721, 5, 3)	concrete (1030, 8, 1)	energy (768, 8, 2)	housing (506, 13, 1)
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0)$	0.0027 ± 0.0001	6.08 ± 0.65	2.21 ± 0.21	$3.56 \pm 1.07^{\dagger}$
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.25)$	0.0027 ± 0.0001	5.79 ± 0.74	1.82 ± 0.18	$3.48 \pm 1.15^{\dagger}$
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.5)$	0.0027 ± 0.0002	5.61 ± 0.65	1.10 ± 0.10	$3.42 \pm 1.04^{\dagger}$
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.75)$	0.0029 ± 0.0002	5.67 ± 0.73	1.04 ± 0.12	$3.43 \pm 1.07^{\dagger}$
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=1.0)$	0.0034 ± 0.0004	$5.55 \pm 0.77^{\dagger}$	1.44 ± 0.38	$3.50 \pm 0.95^{\dagger}$
$\mathcal{L}_{\mathrm{MM}}$	0.0035 ± 0.0005	6.28 ± 0.82	2.38 ± 0.23	4.02 ± 1.18
$\mathcal{L}_{\mathrm{MSE}}$	0.0029 ± 0.0001	$\mathbf{4.96 \pm 0.64}$	$\mathbf{0.95 \pm 0.08}$	$3.24 \pm 1.08^{\dagger}$
Student-t	$\mathbf{0.0026 \pm 0.0000}$	5.82 ± 0.59	2.22 ± 0.18	$3.48 \pm 1.17^{\dagger}$
xVAMP	0.0026 ± 0.0001	$5.44 \pm 0.64^{\dagger}$	2.06 ± 0.19	$3.23 \pm 1.00^{\dagger}$
xVAMP*	$0.0027 \pm 0.0002^{\dagger}$	$5.35 \pm 0.73^{\dagger}$	2.03 ± 0.13	$3.38 \pm 1.15^{\dagger}$
VBEM	0.0037 ± 0.0008	$5.21 \pm 0.58^{\dagger}$	1.36 ± 0.36	$3.32 \pm 1.06^{\dagger}$
VBEM*	0.0026 ± 0.0002	$5.17 \pm 0.59^{\dagger}$	1.26 ± 0.23	$\mathbf{3.19 \pm 1.02}$
Loss	kin8m (8192, 8, 1)	naval (11934, 16, 2)	power (9568, 4, 1)	protein (45730, 9, 1)
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0)$	0.087 ± 0.004	0.0022 ± 0.0006	$4.06 \pm 0.18^{\dagger}$	4.49 ± 0.11
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.25)$	0.083 ± 0.003	0.0012 ± 0.0004	$4.04 \pm 0.18^{\dagger}$	$4.35 \pm 0.05^{\dagger}$
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.5)$	$0.082 \pm 0.003^{\dagger}$	0.0006 ± 0.0002	$4.04 \pm 0.17^{\dagger}$	$4.31 \pm 0.02^{\dagger}$
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.75)$	$0.081 \pm 0.004^{\dagger}$	$0.0004 \pm 0.0001^{\dagger}$	$4.04 \pm 0.15^{\dagger}$	$4.28 \pm 0.02^{\dagger}$
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=1.0)$	$0.081 \pm 0.003^{\dagger}$	$0.0004 \pm 0.0001^{\dagger}$	$4.06 \pm 0.18^{\dagger}$	$4.31 \pm 0.05^{\dagger}$
$\mathcal{L}_{\mathrm{MM}}$	$0.082 \pm 0.003^{\dagger}$	0.0005 ± 0.0001	$4.07 \pm 0.16^{\dagger}$	$4.32 \pm 0.07^{\dagger}$
$\mathcal{L}_{\mathrm{MSE}}$	$\mathbf{0.081 \pm 0.003}$	$\mathbf{0.0003 \pm 0.0001}$	$\mathbf{4.01 \pm 0.19}$	$\mathbf{4.28 \pm 0.07}$
Student-t	0.085 ± 0.005	0.0023 ± 0.0007	$4.02 \pm 0.16^{\dagger}$	4.76 ± 0.24
xVAMP	$0.081 \pm 0.003^{\dagger}$	0.0019 ± 0.0008	$4.03 \pm 0.17^{\dagger}$	$4.38 \pm 0.05^{\dagger}$
xVAMP*	$0.082 \pm 0.003^{\dagger}$	0.0017 ± 0.0006	$4.03 \pm 0.18^{\dagger}$	$4.31 \pm 0.02^{\dagger}$
VBEM	$0.082 \pm 0.003^{\dagger}$	0.0010 ± 0.0005	$4.09 \pm 0.15^{\dagger}$	$4.31 \pm 0.01^{\dagger}$
VBEM*	$0.082 \pm 0.004^{\dagger}$	0.0018 ± 0.0006	$4.02 \pm 0.18^{\dagger}$	$4.35 \pm 0.09^{\dagger}$
Loss	superconductivity (21263, 81, 1)	wine-red (1599, 11, 1)	wine-white (4898, 11, 1)	yacht (308, 6, 1)
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0)$	13.85 ± 0.38	$0.636 \pm 0.038^{\dagger}$	$0.662 \pm 0.007^{\dagger}$	1.22 ± 0.47
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.25)$	13.37 ± 0.24	$0.638 \pm 0.036^{\dagger}$	$0.661 \pm 0.008^{\dagger}$	1.73 ± 1.00
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.5)$	13.19 ± 0.47	$0.635 \pm 0.037^{\dagger}$	$0.662 \pm 0.008^{\dagger}$	2.35 ± 1.44
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=0.75)$	13.25 ± 0.41	$0.638 \pm 0.035^{\dagger}$	$0.662 \pm 0.007^{\dagger}$	1.97 ± 1.03
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta=1.0)$	13.41 ± 0.35	$0.639 \pm 0.035^{\dagger}$	$\mathbf{0.659 \pm 0.010}$	2.08 ± 1.13
$\mathcal{L}_{\mathrm{MM}}$	13.66 ± 0.58	$0.652 \pm 0.044^{\dagger}$	0.665 ± 0.006	3.02 ± 1.38
$\mathcal{L}_{\mathrm{MSE}}$	$\mathbf{12.49 \pm 0.24}$	$0.633 \pm 0.036^{\dagger}$	$0.661 \pm 0.008^{\dagger}$	$0.78 \pm 0.25^{\dagger}$
Student-t	13.81 ± 0.50	$0.636 \pm 0.038^{\dagger}$	$0.662 \pm 0.007^{\dagger}$	1.34 ± 0.63
xVAMP	13.52 ± 0.51	$0.635 \pm 0.035^{\dagger}$	$0.662 \pm 0.012^{\dagger}$	0.99 ± 0.43
xVAMP*	13.44 ± 0.37	$0.633 \pm 0.035^{\dagger}$	$0.661 \pm 0.010^{\dagger}$	1.13 ± 0.66
VBEM	$12.67 \pm 0.48^{\dagger}$	$0.639 \pm 0.041^{\dagger}$	$0.663 \pm 0.012^{\dagger}$	1.66 ± 0.84
VBEM*	13.30 ± 0.32	$\mathbf{0.633 \pm 0.040}$	$0.663 \pm 0.013^{\dagger}$	$\mathbf{0.65 \pm 0.20}$

that mixture. We compare $\beta-$NLL with $\mathcal{L}_{\text{MM}}$, $\mathcal{L}_{\text{NLL}}$ with a fixed variance of 1, Student-t (Takahashi et al., 2018; Detlefsen et al., 2019), xVAMP and VBEM (Stirn & Knowles, 2020).

We evaluate on MNIST and FashionMNIST. Table S3 presents quantitative results. VBEM achieves the best reconstruction error at the expense of poor log-likelihood. Vice-versa, Student-t, xVAMP, xVAMP*, VBEM* achieves strong log-likelihoods but worse reconstruction errors. $\beta-$NLL with $\beta > 0$ provides a good compromise between log-likelihood and reconstruction error. Figure S5 shows qualitative examples. Our β-NLL loss with $\beta > 0$ allows to learn good reconstructions and meaningful uncertainties. Moreover, samples produced from sampling latents from the prior produce semantically meaningful images, indicating that adding our loss function does not break the disentangling properties of VAEs. Compare that to Student-t, xVAMP(*) and VBEM(*), which do not produce similarly clear images when sampling from the prior.

Table S3: Results for generative modelling with Variational Autoencoders on MNIST and Fashion-MNIST. We report RMSE and posterior predictive log-likelihood (LL) with standard deviation over 5 independent trials.

	MNIST		**Fashion-MNIST**	
Loss	RMSE ↓	LL ↑	RMSE ↓	LL ↑
$\mathcal{L}_{\text{NLL}}$ ($\sigma^2 = 1$)	0.153 ± 0.002	-730 ± 0	0.143 ± 0.002	-729 ± 0
$\mathcal{L}_{\beta-\text{NLL}}(\beta = 0)$	0.237 ± 0.002	2116 ± 55	0.170 ± 0.001	1940 ± 104
$\mathcal{L}_{\beta-\text{NLL}}(\beta = 0.25)$	0.181 ± 0.004	2511 ± 88	0.140 ± 0.002	2010 ± 39
$\mathcal{L}_{\beta-\text{NLL}}(\beta = 0.5)$	0.151 ± 0.003	2220 ± 25	0.125 ± 0.003	1639 ± 52
$\mathcal{L}_{\beta-\text{NLL}}(\beta = 0.75)$	0.142 ± 0.001	1954 ± 50	0.131 ± 0.001	1331 ± 32
$\mathcal{L}_{\beta-\text{NLL}}(\beta = 1.0)$	0.152 ± 0.001	1706 ± 30	0.138 ± 0.002	1142 ± 26
$\mathcal{L}_{\text{MM}}$	0.260 ± 0.000	385 ± 11	0.295 ± 0.000	-151 ± 4
Student-t	0.273 ± 0.002	4291 ± 103	0.182 ± 0.002	2857 ± 9
xVAMP	0.225 ± 0.002	2989 ± 268	0.161 ± 0.001	2158 ± 100
xVAMP*	0.225 ± 0.001	3062 ± 215	0.160 ± 0.002	2150 ± 131
VBEM	0.114 ± 0.001	719 ± 9	0.108 ± 0.000	660 ± 2
VBEM*	0.176 ± 0.008	3213 ± 238	0.150 ± 0.003	2244 ± 78

A.5 Depth Regression

We evaluate $\beta-$NLL on the task of depth regression on the NYUv2 dataset (Silberman et al., 2012). For this purpose, we use a state-of-the-art method for depth regression, AdaBins (Bhat et al., 2021), and train it with different loss function. Note that we remove the Mini-ViT Transformer module from the model, and thus our results are not directly comparable with those reported by (Bhat et al., 2021).

Table S4 presents quantitative results. $\beta-$NLL with $\beta > 0$ achieves better RMSE than the NLL loss. $\beta-$NLL with $\beta = 0.5$ again provides a good trade-off, achieving similar RMSE as $\mathcal{L}_{\text{MSE}}$ and performing better on some of the other metrics. Figure S6 shows qualitative examples. Compared to $\mathcal{L}_{\text{NLL}}$, the depth maps predicted by $\beta-$NLL with $\beta > 0$ are noticeably sharper.

B Datasets and Training Settings

Sinusoidal without heteroscedastic noise This dataset is created by taking 1 000 uniformly spaced points on the interval $[0, 12]$ as inputs x and applying the function $y(x) = 0.4 \sin(2\pi x) + \xi$ to them to create the targets y, where ξ is Gaussian noise with a standard deviation of 0.01.

Sinusoidal heteroscedastic noise We use the synthetic data as introduced in Detlefsen et al. (2019). From the functional form $y = x \sin(x) + x\xi_1 + \xi_2$, with Gaussian noise with standard deviation $\sigma = 0.3$ for ξ_1 and ξ_2, we sample 500 points uniformly spaced in the interval $[0, 10]$. The model is a MLP with one hidden layer with 50 units and $\tanh$ activation function (as used in Detlefsen et al. (2019) and Stirn & Knowles (2020)).

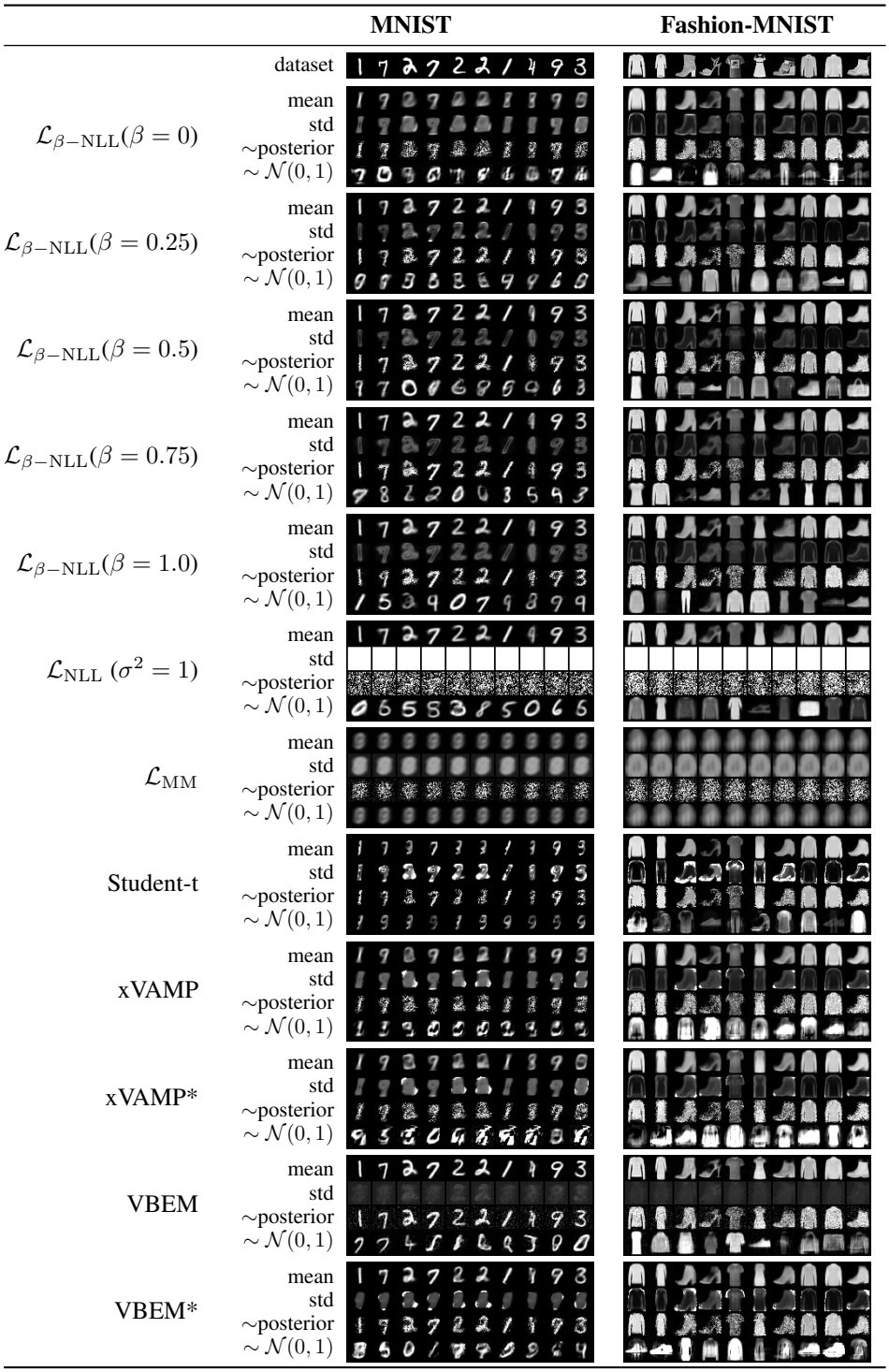

Figure S5: Generative modelling with Variational Autoencoders on MNIST and Fashion-MNIST. The overall first row shows inputs from the test set. For each method, we present posterior predictive means (i.e. reconstructions), the posterior predictive standard deviations, samples from the posterior predictive distribution, and finally samples using the prior $\mathcal{N}(0, 1)$. $\mathcal{L}_{\text{NLL}}(\sigma^2 = 1)$ refers to Gaussian log-likelihood with a fixed variance of 1. Values are clipped to the interval $[0, 1]$. Examples are not cherry-picked.

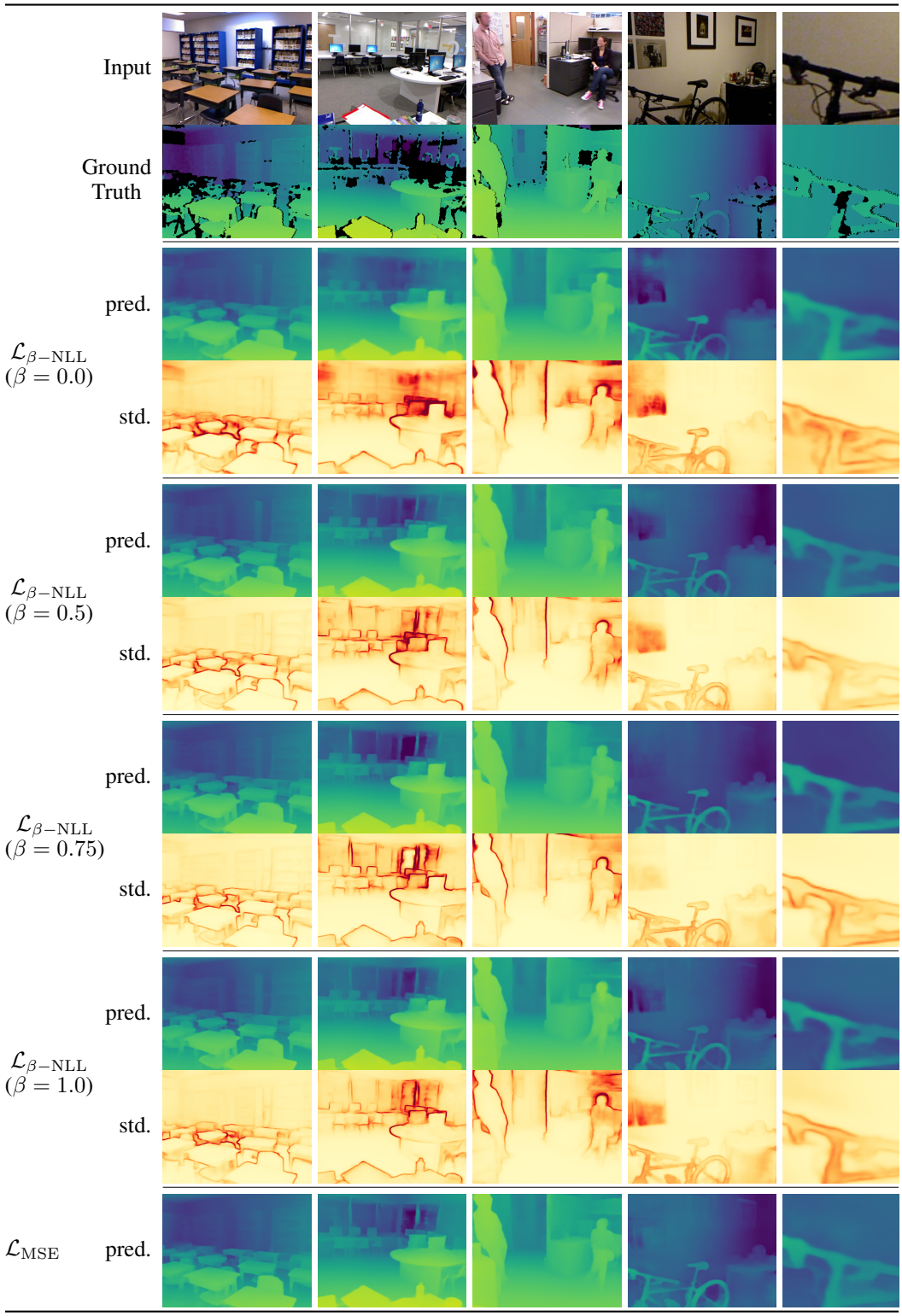

Figure S6: Example results for depth regression on the NYUv2 dataset (Silberman et al., 2012). First two rows show input image and ground truth depth map, where black values in the depth map represent missing values. For each method, we present the predicted depth map (pred.), and the aleatoric uncertainty in form of the predicted standard deviation (std.). Results for $\mathcal{L}_{\beta-\mathrm{NLL}}$ with $\beta > 0$ are noticeably sharper. The last column shows a magnified view on the previous picture.

Table S4: Results for depth regression on the NYUv2 dataset (Silberman et al., 2012). We adapt a state-of-the-art network for depth regression from AdaBins (Bhat et al., 2021) and train it with different loss functions. In contrast to AdaBins, our network does not include the Mini-ViT Transformer module and thus our results are not directly comparable to AdaBins'. For reference, we also report numbers from other recent literature on this task. Notably, the Gaussian NLL in all variants clearly outperforms the Scale Invariant (SI) loss, even though it is a loss function specifically designed for the task of depth regression. We refer to Bhat et al. (2021) for a description of the metrics.

Method	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	REL $\downarrow$	RMSE $\downarrow$	$\log_{10} \downarrow$	LL $\uparrow$
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0)$	0.8855	0.9796	0.9959	0.1094	0.3854	0.0462	-4.52
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.25)$	0.8887	0.9812	0.9956	0.1081	0.3818	0.0458	-8.15
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.5)$	0.8885	0.9813	0.9956	0.1093	0.3789	0.0458	-7.50
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 0.75)$	0.8902	0.9804	0.9952	0.1095	0.3800	0.0462	-7.35
$\mathcal{L}_{\beta-\mathrm{NLL}}(\beta = 1.0)$	0.8872	0.9813	0.9958	0.1088	0.3845	0.0467	-5.10
$\mathcal{L}_{\mathrm{MSE}}$	0.8890	0.9806	0.9960	0.1086	0.3776	0.0461	—
$\mathcal{L}_1$	0.8877	0.9798	0.9955	0.1073	0.3850	0.0459	—
SI Loss (Bhat et al., 2021)	0.881	0.980	0.996	0.111	0.419	—	—
AdaBins (Bhat et al., 2021)	0.903	0.984	0.997	0.103	0.364	0.044	—
BTS (Lee et al., 2019)	0.885	0.978	0.994	0.110	0.392	0.047	—
DAV (Chen et al., 2019)	0.882	0.980	0.996	0.108	0.412	—	—

UCI Datasets We use the UCI datasets suite commonly used to benchmark uncertainty estimation, stemming from the UCI Machine Learning Repository[3]. In particular, we use the training-test protocol from (Hernández-Lobato & Adams, 2015; Gal & Ghahramani, 2016), and their data splits [4].

Inputs and targets are whitened on the training set. Metrics are reported in the original scale of the data. Each dataset is divided into 20 randomly sampled train-test splits (80%-20%). For each split, we further divide the training set into 80% training data and 20% validation data and search for an optimal learning rate from the set $\{10^{-4}, 3 \cdot 10^{-4}, 7 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}, 7 \cdot 10^{-3}\}$ by monitoring log-likelihood on the validation set. We train for a maximum of 20 000 updates, besides on the larger "kin8m", "power plant", "protein" and "naval" datasets, where we train for a maximum of 100 000 updates. We perform early stopping with a patience of 50 epochs, retrain the model with the best found learning rate on the full training set, and then evaluate on the test split. The reported performance and standard deviations are taken as averages over all test splits. Note that the performance we report is not comparable with other publications, as performance is known to differ strongly over different data splits. Some other works also perform early-stopping on the test set, which distorts the results.

We use a single-layer relu hidden network with 50 neurons, aside from "protein", where we use 100 neurons. The batch size is 256.

Following Stirn & Knowles (2020), for each method, we report the number of datasets for which the method statistically indistinguishable from the respective best method in Table 1 ("Ties"). For this purpose, we performed a two-sided Kolmogorov-Smirnov with a $p \leq 0.05$ significance level.

ObjectSlide This environment consists of an agent whose task it is to slide an object to a target location (Seitzer et al., 2021). The continuous state space consists of 4 dimensions: agent and object position and velocity, and the continuous action space is a one-dimensional movement command. The forward prediction task consists of predicting the change in object position in the next state from current state and action. The dataset we use consists of 180 000 transitions collected using a random policy, which we split into training, validation and testing sets with 60 000 transitions each. Inputs and targets are whitened on the training set. Metrics are reported in the original scale of the data. We train for a maximum of 5 000 epochs with a batch size of 256, and evaluate the model with the best validation log-likelihood on the test set afterwards.

[3] https://archive.ics.uci.edu

[4] available under https://github.com/yaringal/DropoutUncertaintyExps

Fetch-PickAndPlace We use the Fetch-PickAndPlace environment (Plappert et al., 2018) from OpenAI Gym (Brockman et al., 2016) as a challenging real-world scenario. The task of the agent is to use position-controlled 7 DoF robotic arm to lift an object to a target location in space. The state space is 25-dimensional and the action space is 4-dimensional. As in ObjectSlide, the prediction task is to predict the 3D-dimensional change in object position from the current state and action. We use 840 000 transitions collected using the APEX method (Pinneri et al., 2021) as our dataset, which we split into 70% training, 15% validation and 15% testing data. Inputs and targets are whitened on the training set. Metrics are reported in the original scale of the data. We train for a maximum of 500 epochs with a batch size of 256, and evaluate the model with the best validation log-likelihood on the test set afterwards.

NYUv2 Depth Regression We use the dataset in the variant provided by Lee et al. (2019) [5].

C Hyperparameter Settings and Implementation Details

For all experiments, we used the Adam optimizer (Kingma & Ba, 2015) with standard settings $\beta_1 = 0.9, \beta_2 = 0.999$. We parametrize the Gaussian distribution using two linear layers on top of shared features produced by a MLP. The variance $\hat{\sigma}^2(x)$ is constrained to the positive region using the $\mathrm{softplus}(x) = \log(1 + \exp(x))$ activation function. We additionally add a small constant of 10^{-8} to prevent the variance from collapsing to zero, and clamp the maximum variance to 1 000.

C.1 Sinusoidal Regression Problem

The sinusoidal fit in Fig. 1 results from a network with two hidden layers and 128 neurons each, $\tanh$ activation, and optimized with learning rate $5 \cdot 10^{-4}$ and a batch size of 100. For the experiment in Sec. 5.1, we scan over learning rates and architectures with different hidden layers and units per layer, as detailed in Table S5.

Table S5: Architectures used for the sinusoidal regression task. Fully connected feed-forward neural networks with $\tanh$ activation function.

Architecture #	0	1	2	3	4
# Hidden Layers	2	2	2	3	3
# Units per Layer	32	64	128	128	256

C.2 ObjectSlide and Fetch-PickAndPlace

For each tested loss function, we performed a grid search for the ObjectSlide and Fetch-PickAndPlace datasets, We report the parameters in Table S6. For the results in Table 2, we retrained the model configurations with the best validation log-likelihood on the grid search, and evaluated them on the hold-out test set.

The best found configuration on ObjectSlide coincided to 3 hidden layers with 128 neurons, relu activation and learning rate 0.001. The best found configuration on Fetch-PickAndPlace coincided to 4 hidden layers with 128 neurons, relu activation and learning rates 0.0003 for $\mathcal{L}_{\mathrm{NLL}}$, $\mathcal{L}_{\beta-\mathrm{NLL}}$ with $\beta = 0.25$ and $\beta = 0.5$, and learning rate 0.001 for $\mathcal{L}_{\mathrm{MSE}}$, $\mathcal{L}_{\mathrm{MM}}$, $\mathcal{L}_{\beta-\mathrm{NLL}}$ with $\beta = 0.75$ and $\beta = 1.0$.

C.3 Variational Autoencoders

We largely follow Stirn & Knowles (2020) for their training settings for the VAE experiment. In particular, we use an encoder with three layers of $512, 256, 128$ neurons, a decoder with three layers of $128, 256, 512$ neurons and ReLU activations. The latent space is 10-dimensional for MNIST and 25-dimensional for FashionMNIST. We train the VAEs for a maximum of 1 000 epochs, using Adam with a learning rate of 0.0003 and a batch size of 256. Early stopping with a patience of 50 epochs

[5] available under https://github.com/cogaplex-bts/bts

Table S6: Hyperparameter settings for grid search on ObjectSlide and Fetch-PickAndPlace datasets. We run 96 configurations per loss function.

Hyperparameter	Set of Values
Learning Rate	$\{3 \cdot 10^{-5}, 10^{-4}, 3 \cdot 10^{-4}, \cdot 10^{-3}\}$
# Hidden Layers	$\{2, 3, 4\}$
# Units per Layer	$\{128, 256, 386, 512\}$
Activation	$\{\tanh, \mathrm{relu}\}$

is performed on the log-likelihood of the validation set. The validation set consists of 20% of the MNIST/FashionMNIST training set.

C.4 Depth Regression

We use the official implementation of AdaBins (Bhat et al., 2021)[6], thereby reproducing their exact training settings and evaluation protocol. We remove the AdaBins/mini-ViT Transformer from the model. Instead, the feature map output by the U-Net is reduced to two channels using a 1×1 convolution, where we use the first channel as the mean prediction and the second channels as the variance prediction. In this case, both mean and variance are constrained to positive numbers by a softplus activation. On top of that, we add a positive offset to ensure a minimum output value of 10^{-3} for the mean (the minimum possible depth value) and 10^{-6} for the variance, and clamp both mean and variance to a maximum value of 10.

C.5 Implementation of beta-NLL in PyTorch

```
def beta_nll_loss(mean, variance, target, beta):
    """Compute beta-NLL loss

    :param mean: Predicted mean of shape B x D
    :param variance: Predicted variance of shape B x D
    :param target: Target of shape B x D
    :returns: Loss per batch element of shape B
    """
    loss = 0.5 * ((target - mean) ** 2 / variance + variance.log())

    if beta > 0:
        loss = loss * variance.detach() ** beta

    return loss.sum(axis=-1)
```

[6] https://github.com/shariqfarooq123/AdaBins