Supplementary Material for SDE-Net: Equipping Deep Neural Networks with Uncertainty Estimates

1. Proof of Theorem 1

Theorem 1 can be seen as a special case of the existence and uniqueness theorem of a general stochastic differential equation. The following derivation is adapted from (Lalley, 2016). To prove Theorem 1, we first introduce two lemmas.

Lemma 1. *Let $y(t)$ be a nonnegative function that satisfies the following condition: for some $T \leq \infty$, there exist constants $A, B \geq 0$ such that:*

$$y(t) \leq A + B\int_0^t y(s)ds < \infty \quad \textit{for all} \quad 0 \leq t \leq T. \tag{1}$$

Then

$$y(t) \leq Ae^{Bt} \quad \textit{for all} \quad 0 \leq t \leq T. \tag{2}$$

Proof. W.l.o.g., we assume that $C = \int_0^T y(s)ds < \infty$ and that $T < \infty$. Then, we can obtain that $y(t)$ is bounded by $D \equiv A + BC$ in the interval $[0, T]$. By iterating over inequality (1), we have:

$$\begin{aligned} y(t) &\leq A + B\int_0^t y(s)ds \\ &\leq A + B\int_0^t (A+B)\int_0^s y(r)drds \\ &\leq A + BAt + B^2\int_0^t\int_0^s (A + B\int_0^r y(q)dq)drds \\ &\leq A + BAt + B^2At^2/2! + B^3\int_0^t\int_0^s\int_0^r (A + B\int_0^q y(p)dp)dqdrds \\ &\leq \cdots . \end{aligned} \tag{3}$$

After k iterations, the first k terms are the series for Ae^{Bt}. The last term is a $(k+1)$-fold iterated integral I_k. Because $y(t) \leq D$ in the interval $[0, T]$, the integral I_k is bounded by $B^kDt^{k+1}/(k+1)!$. This converges to zero uniformly for $t \leq T$ as $k \to \infty$. Hence, inequality (2) follows.

□

Lemma 2. *Let $y_n(t)$ be a sequence of nonnegative functions such that for some constants $B, C < \infty$,*

$$\begin{aligned} y_0(t) &\leq C \quad \textit{for all} \quad t \leq T \quad \textit{and} \\ y_{n+1}(t) &\leq B\int_0^t y_n(s)ds < \infty \quad \textit{for all} \quad t \leq T \quad \textit{and} \quad n = 0, 1, 2, \cdots . \end{aligned} \tag{4}$$

Then,

$$y_n(t) \leq CB^nt^n/n! \quad \textit{for all} \quad t \leq T. \tag{5}$$

Proof.

$$\begin{aligned} &y_1(t) \le B\int_0^t C ds = BCt \\ &y_2(t) \le B\int_0^t BCs ds = CB^2t^2/2! \\ &y_3(t) \le B\int_0^t CB^2s^2/2 ds = CB^3t^3/3! \\ &\cdots. \end{aligned} \tag{6}$$

After n iterations, we have $y_n(t) \le CB^nt^n/n!$ for all $t \le T$. □

Suppose that for some initial value $\boldsymbol{x}_0$ there are two different solutions:

$$\begin{aligned} \boldsymbol{x}_t &= \boldsymbol{x}_0 + \int_0^t f(\boldsymbol{x}_s, s; \boldsymbol{\theta}_f) ds + \int_0^t g(\boldsymbol{x}_0; \boldsymbol{\theta}_g) dW_s \quad \text{and} \\ \boldsymbol{y}_t &= \boldsymbol{x}_0 + \int_0^t f(\boldsymbol{y}_s, s; \boldsymbol{\theta}_f) ds + \int_0^t g(\boldsymbol{x}_0; \boldsymbol{\theta}_g) dW_s. \end{aligned} \tag{7}$$

Since the diffusion net g is uniformly Lipschitz, $\int_0^t g(\boldsymbol{x_0}; \boldsymbol{\theta}_g) dW_s$ is bounded in compact time intervals. Then, we substract these two solutions and get:

$$\boldsymbol{x}_t - \boldsymbol{y}_t = \int_0^t (f(\boldsymbol{x}_s, s; \boldsymbol{\theta}_f) - f(\boldsymbol{y}_s, s; \boldsymbol{\theta}_f)) ds. \tag{8}$$

Since the drift net f is uniformly Lipschitz, we have that for some constant $B < \infty$,

$$|\boldsymbol{y}_t - \boldsymbol{x}_t| \le B\int_0^t |\boldsymbol{y}_s - \boldsymbol{x}_s| ds \quad \text{for all} \quad t < \infty. \tag{9}$$

It is obvious that $\boldsymbol{y}_t - \boldsymbol{x}_t \equiv 0$ from Lemma 1 by letting $A = 0$. Thus, the stochastic differential equation has at most one solution for any particular initial value $\boldsymbol{x}_0$.

Then, we prove the the existence of the solutions. For a fix initial value $\boldsymbol{x}_0$, we define a sequence of adapted process $\boldsymbol{x}_n(t)$ by:

$$\boldsymbol{x}_{n+1}(t) = \boldsymbol{x}_0 + \int_0^t f(\boldsymbol{x}_n(s), s; \boldsymbol{\theta}_f) ds + g(\boldsymbol{x_0}; \boldsymbol{\theta}_g) W_t \tag{10}$$

The processes $\boldsymbol{x}_{n+1}(t)$ are well-defined and have continuous paths, by induction on n. Because the drift net f is Lipschitz, we have:

$$|\boldsymbol{x}_{n+1}(t) - \boldsymbol{x}_n(t)| \le B\int_0^t |\boldsymbol{x}_n(s) - \boldsymbol{x}_{n-1}(s)| ds. \tag{11}$$

Therefore, Lemma 2 implies that for any $T < \infty$,

$$|\boldsymbol{x}_{n+1}(t) - \boldsymbol{x}_n(t)| \le CB^nT^n/n! \quad \text{for all} \quad t \le T. \tag{12}$$

It follows that the processes $\boldsymbol{x}_n(t)$ converge uniformly in compact time intervals $[0, T]$; thus the limit process $\boldsymbol{x}(t)$ has continuous trajectories according to the dominated convergence theorem and the continuity of f.

2. Experimental Details

2.1. Classification Setup Details

Data preprocessing. As PN and SDE-Net both require OOD samples during the training process, we perturb training data by Gaussian noise as pseudo OOD data by default. On both MNIST and SVHN, the mean of the Gaussian noise is set to zero and the variance is set to 4.

We have also experimented with using external data as OOD data for model training or test, which requires re-scaling external data to match the target dataset. Specifically, for the classification task on MNIST, we used SEMEION and upscaled the images to 28×28; we also tried CIFAR10 and transformed images into greyscale and downsampled them to 28×28 size.

Model hyperparameters. we use one SDE-Net block in replace of 6 residual blocks and set the number of subintervals as $N = 6$ for fair comparison. We perform one forward propagation during training time and 10 forward propagations at test time. To make the training procedure more stable, we use a smaller value of $\sigma_{\max}$ during training. Specifically, we set $\sigma_{\max} = 1$ during trainining and $\sigma_{\max} = 50$ at test time for both MNIST and SVHN.

The dropout rate for MC-dropout is set to 0.1 as in (Lakshminarayanan et al., 2017) (we also tested 0.5, but that setting performed worse). For DeepEnsemble, we use 5 ResNets in the ensemble. For PN, we set the concentration parameter to 1000 for both MNIST and SVHN as suggested in the original paper. We use the standard normal prior for both BBP and p-SGLD. The variances of the prior are set to 0.1 for BBP and 0.01 for p-SGLD to ensure convergence. We use 50 posterior samples for MC-dropout, BBP and p-SGLD at test time.

For PGD attack, we set the perturbations size ϵ to 0.3 $(16/255)$ and step size to $2/255$ $(0.4/255)$ on MNIST (SVHN).

Model optimization. On the MNIST dataset, we use the stochastic gradient descent algorithm with momentum 0.9, weight decay 5×10^{-4}, and mini-batch size 128. BBP and p-SGLD are trained with 200 epochs to ensure convergence while other methods are trained with 40 epochs. The initial learning rate is set to 0.1 for for drift network, MC-dropout and DeepEnsemble while 0.01 for PN. It then decreased at epoch 10, 20 and 30. The learning rate for drift network is initially set to 0.01 and then decreased at epoch 15 and 30. The learning rate for BBP is initially set to 0.001 and then decreased at epoch 80 and 160. We use an initial learning rate 0.0001 for p-SGLD and then decreased it at epoch 50. The decrease rate for SGD learning rate is set to 0.1.

On the SVHN dataset, we again use the stochastic gradient descent algorithm with momentum 0.9 and weight decay 5×10^{-4}. BBP and p-SGLD are trained with 200 epochs to ensure convergence while other methods are trained with 60 epochs. The initial learning rate is set to 0.1 for for drift network, MC-dropout and DeepEnsemble while 0.01 for PN. It then decreased at epoch 20 and 40. The learning rate for diffusion network is set as 0.005 initially and then decreased at epoch 10 and 30. p-SGLD uses a contant learning rate 0.0001. The learning rate for BBP is initially set to 0.001 and then decreased at epoch 80 and 160.

2.2. Regression Setup Details

Data preprocessing. We normalize both the features and targets (0 mean and 1 variance) for the regression task. We repeat the features of Boston Housing data 6 times and pad zeroes for the remaining entries to make the number of features of the two datasets equal. We perturb training data by Gaussian noise (zero mean and variance 4) as pseudo OOD data.

Model hyperparameters. The neural net used in the baselines has 6-hidden layers with ReLU nonlinearity. For fair comparison, we set the number of subintervals as 4 and then place two layers before and after the SDE-Net block respectively. The dropout rate for MC-dropout is set to 0.05 as in (Gal & Ghahramani, 2016). We set $\sigma_{\max}$ to 0.1 initially and increase it to 0.5 at epoch 40. During training, we only perform 1 forward pass. The number of stochastic forward passes is 10 for SDE-Net at test time. 20 posterior samples are used for MC-dropout, BBP and p-SGLD at test time. The variance is set to 0.1 for both BBP and p-SGLD to ensure convergence.

Model optimization. We use the stochastic gradient descent algorithm with momentum 0.9, weight decay 5×10^{-4}, and mini-batch size 128. The number of training epochs is 60. The learning rate for drift net is initially set to 0.0001 and then deceased at epoch 20. The learning rate for the diffusion net is set to 0.01. The learning rate for BBP and p-SGLD is initially set to 0.01 and then decreased at epoch 20. The learning rate for other baselines is initially set to 0.001 and then decreased at epoch 20.

2.3. Active Learning Setup

Data preprocessing. We normalize both the features and targets (0 mean and 1 variance) for the active learning task. We randomly select 50 samples from the original training set as the starting point.

Model hyperparameters. The network architecture and model hyperparameters are the same as those we used in the OOD detection task for regression.

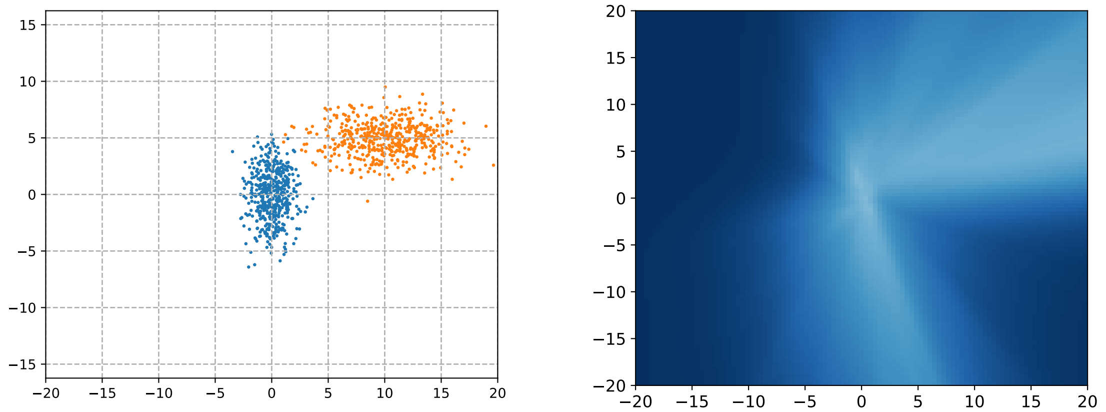
Model optimization. We use the stochastic gradient descent algorithm with momentum 0.9, weight decay 5×10^{-4}, and mini-batch size 50. The number of training epochs is 100. The learning rate for drift net and baselines is set to 0.0001. The learning rate for the diffusion net is set to 0.01.

3. Additional Experiments

3.1. Visulization Using Synthetic Dataset

In this subsection, we demonstrate the capability of SDE-Net of obtaining meaningful epistemic uncertainties. For this purpose, we generate a synthetic dataset from a mixture of two Gaussians. Then, we train the SDE-Net on this toy dataset. Both the drift neural network and diffusion network have one hidden layer with ReLU activation .

Figure 1b shows the uncertainty obtained by SDE-Net. Specifically, it visualizes the epistemic uncertainty given by the variance of the Brownian motion term. As we can see, the uncertainty is low in the region covered by the training data while high outside the training distribution.



(a) Training data distribution.

(b) Epistemic Uncertainty estimated by SDE-Net.

Figure 1. Visualization of the epistemic uncertainty estimated by SDE-Net (darker colors represent higher uncertainties in the heat map).

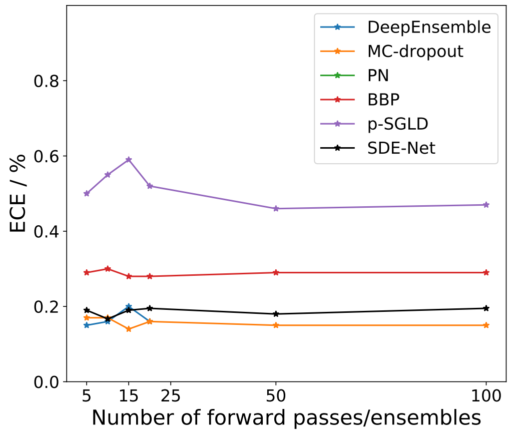
3.2. Expected Calibration Error

In this subsection, we measure the expected calibration error (ECE, (Guo et al., 2017)) to see if the confidences produced by the models are trustworthy. Fig. 2 shows the ECE of each method on MNIST and SVHN. On MNIST, SDE-Net can achieve competitive results compared with DeepEnsemble and MC-dropout and outperforms other methods. On SVHN, SDE-Net outperforms all the baselines.

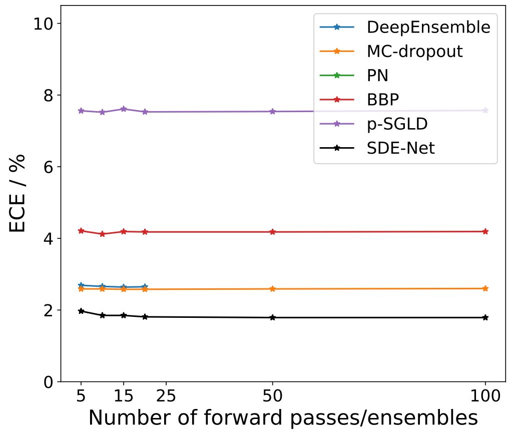
3.3. Ablation Study

Robustness to different pseudo OOD data. In this set of experiments, we report additional experimental results for OOD detection in classification tasks. We use MNIST as the in-distribution training dataset, and explore using other data sources as OOD data beyond using in-distribution data perturbed by Gaussian noise. The results are shown in Table 1. As we can see, the performance of PN is very poor when using Gaussian noise and training data perturbed by Gaussian noise. When using SVHN as OOD data during training, its performance is good. This suggests that PN is easy to be overfitted by the OOD data used in training. Our SDE-Net can achieve good performance in all settings, which shows its superior robustness.

Is the OOD regularizer necessary? Our loss objective includes an OOD regularization term which allows us to explicitly train the epistemic uncertainty for each data point. This regularizer can be interpreted as our parameter belief from the data space. That is we want our model to give uncertain outputs for OOD data. To verify the necessity of this regularization term, we test the uncertainty estimates of SDE-Net trained without the regularizer. As we can see from Table. 2, the performance



(a) MNIST



(b) SVHN

Figure 2. Expected calibration error (ECE) vs number of forward passes/ensembles. PN is outside the range and not shown

Table 1. Additional Results for OOD detection. MNIST is used as in-distribution training data. The OOD data used during training is in the bracket beside each model. Gaussian means directly sampling from $\mathcal{N}(0,1)$ as pseudo OOD data. Training+Gaussian means perturbing training data by Gaussian noise (0 mean and variance 4) as pseudo OOD data. SVHN means directly use the training set of SVHN as pseudo OOD data. We report the average performance and standard deviation for 5 random initializations.

OOD Data (test)	Model	TNR at TPR 95%	AUROC	Detection accuracy	AUPR in	AUPR out
SVHN	SDE-Net(SVHN)	99.9 ± 0.0	99.9 ± 0.0	99.8 ± 0.1	99.9 ± 0.0	99.9 ± 0.0
	SDE-Net(Gaussian)	99.4 ± 0.1	99.9 ± 0.0	98.5 ± 0.2	99.7 ± 0.1	100.0 ± 0.0
	SDE-Net(training+Gaussian)	97.8 ± 1.1	99.5 ± 0.2	97.0 ± 0.2	98.6 ± 0.6	99.8 ± 0.1
	PN(SVHN)	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	PN(Gaussian)	89.0 ± 2.9	92.9 ± 1.2	92.3 ± 2.2	68.1 ± 6.5	97.6 ± 0.7
	PN(training+Gaussian)	90.4 ± 2.8	94.1 ± 2.2	93.0 ± 1.4	73.2 ± 7.3	98.0 ± 0.6
SEMEION	SDE-Net(SVHN)	100.0 ± 0.0	99.9 ± 0.0	99.9 ± 0.0	100.0 ± 0.0	99.0 ± 0.2
	SDE-Net(Gaussian)	99.9 ± 0.1	100.0 ± 0.0	99.0 ± 0.3	100.0 ± 0.0	99.8 ± 0.1
	SDE-Net(training+Gaussian)	99.6 ± 0.2	99.9 ± 0.1	98.6 ± 0.5	100.0 ± 0.0	99.5 ± 0.3
	PN(SVHN)	98.0 ± 0.8	98.7 ± 0.3	97.3 ± 1.2	99.6 ± 0.1	95.7 ± 2.3
	PN(Gaussian)	91.0 ± 2.3	94.9 ± 2.6	93.2 ± 1.5	97.8 ± 0.6	86.5 ± 3.5
	PN(training+Gaussian)	93.4 ± 2.2	96.1 ± 1.2	94.5 ± 1.1	98.4 ± 0.7	88.5 ± 1.3
CIFAR10	SDE-Net(SVHN)	100.0 ± 0.0	99.9 ± 0.0	99.7 ± 0.1	99.9 ± 0.1	99.8 ± 0.1
	SDE-Net(Gaussian)	99.8 ± 0.1	100.0 ± 0.0	98.9 ± 0.4	100.0 ± 0.0	100.0 ± 0.0
	SDE-Net(training+Gaussian)	99.7 ± 0.2	99.9 ± 0.0	98.3 ± 0.4	99.9 ± 0.0	99.9 ± 0.0
	PN(SVHN)	100.0 ± 0.0	100.0 ± 0.0	99.8 ± 0.1	100.0 ± 0.0	100.0 ± 0.0
	PN(Gaussian)	96.8 ± 1.2	97.7 ± 0.7	96.5 ± 0.6	94.3 ± 1.2	98.2 ± 0.3
	PN(training+Gaussian)	97.6 ± 0.7	98.3 ± 0.8	97.0 ± 1.2	96.0 ± 1.7	97.3 ± 1.2

of SDE-Net deteriorates to the same level of traditional NNs without the regularizer term. In Bayesian neural network, the principle of Bayesian inference implicitly enables larger uncertainty in the region that lacks training data. Such inference can be costly and we choose to view the DNNs as stochastic dynamic systems. The benefit of such design is that we can directly model the epistemic uncertainty level for each data point by the variance of the Brownian motion.

3.4. Full Results of Table. 2 and Table. 3 of the main paper

Table. 3 shows the full results of Table. 2 of the main paper.

Table 2. Classification and out-of-distribution detection results on MNIST and SVHN. All values are in percentage, and larger values indicates better detection performance. We report the average performance and standard deviation for 5 random initializations.

ID	OOD	Model	TNR at TPR 95%	AUROC	Detection accuracy	AUPR in	AUPR out
MNIST	SEMEION	SDE-Net w.o. reg	93.7 ± 1.1	97.9 ± 0.4	95.2 ± 0.9	99.8 ± 0.1	89.8 ± 1.2
		SDE-Net	$\mathbf{99.6} \pm 0.2$	$\mathbf{99.9} \pm 0.1$	$\mathbf{98.6} \pm 0.5$	$\mathbf{100.0} \pm 0$	$\mathbf{99.5} \pm 0.3$
MNIST	SVHN	SDE-Net w.o. reg	90.3 ± 1.3	96.6 ± 1.3	92.2 ± 1.2	90.0 ± 2.2	98.2 ± 0.4
		SDE-Net	$\mathbf{97.8} \pm 1.1$	$\mathbf{99.5} \pm 0.2$	$\mathbf{97.0} \pm 0.2$	$\mathbf{98.6} \pm 0.6$	$\mathbf{99.8} \pm 0.1$
SVHN	CIFAR10	SDE-Net w.o. reg	68.2 ± 2.4	93.9 ± 0.7	90.3 ± 0.9	97.2 ± 0.7	85.2 ± 1.2
		SDE-Net	$\mathbf{87.5} \pm 2.8$	$\mathbf{97.8} \pm 0.4$	$\mathbf{92.7} \pm 0.7$	$\mathbf{99.2} \pm 0.2$	$\mathbf{93.7} \pm 0.9$
SVHN	CIFAR100	SDE-Net w.o. reg	65.2 ± 1.3	92.9 ± 0.9	88.7 ± 0.6	97.2 ± 0.3	83.4 ± 0.7
		SDE-Net	$\mathbf{83.4} \pm 3.6$	$\mathbf{97.0} \pm 0.4$	$\mathbf{91.6} \pm 0.7$	$\mathbf{98.8} \pm 0.1$	$\mathbf{92.3} \pm 1.1$

Table. 4 shows the full results of Table. 3 of the main paper.

Table 3. Out-of-distribution detection for regression on Year Prediction MSD + Boston Housing. We report the average performance and standard deviation for 5 random initializations.

Model	# Parameters	RMSE	TNR at TPR 95%	AUROC	Detection accuracy	AUPR in	AUPR out
DeepEnsemble	14.9K×5	$\mathbf{8.6}\pm$ NA	$10.9\pm$ NA	$59.8\pm$ NA	61.4±NA	99.3±NA	$1.3\pm$ NA
MC-dropout	14.9K	8.7 ± 0.0	9.6 ± 0.4	53.0 ± 1.2	55.6 ± 1.2	99.2 ± 0.1	1.1 ± 0.1
BBP	30.0K	9.5 ± 0.2	8.7 ± 1.5	56.8 ± 0.9	58.3 ± 2.1	99.0 ± 0.0	1.3 ± 0.1
p-SGLD	14.9K	9.3 ± 0.1	9.2 ± 1.5	52.3 ± 0.7	57.3 ± 1.9	99.4 ± 0.0	1.1 ± 0.2
SDE-Net	12.4K	8.7 ± 0.1	$\mathbf{60.4} \pm 3.7$	$\mathbf{84.4} \pm 1.0$	$\mathbf{80.0} \pm 0.9$	$\mathbf{99.7} \pm 0.0$	$\mathbf{21.3} \pm 4.1$

Table 4. Misclassification detection performance on MNIST and SVHN. We report the average performance and standard deviation for 5 random initializations.

Data	Model	TNR at TPR 95%	AUROC	Detection accuracy	AUPR succ	AUPR err
MNIST	Threshold	85.4 ± 2.8	94.3 ± 0.9	92.1 ± 1.5	99.8 ± 0.1	31.9 ± 8.3
	DeepEnsemble	89.6±NA	$\mathbf{97.5}\pm$ NA	93.2±NA	$\mathbf{100.0}\pm$ NA	$41.4\pm$ NA
	MC-dropout	85.4 ± 4.5	95.8 ± 1.3	91.5 ± 2.2	99.9 ± 0.0	33.0 ± 6.7
	PN	85.4 ± 2.8	91.8 ± 0.7	91.0 ± 1.1	99.8 ± 0.0	33.4 ± 4.6
	BBP	88.7 ± 0.9	96.5 ± 2.1	93.1 ± 0.5	$\mathbf{100.0} \pm 0.0$	35.4 ± 3.2
	P-SGLD	$\mathbf{93.2} \pm 2.5$	96.4 ± 1.7	$\mathbf{98.4} \pm 0.2$	$\mathbf{100.0 \pm} 0.0$	$\mathbf{42.0 \pm} 2.4$
	SDE-Net	88.5 ± 1.3	96.8 ± 0.9	92.9 ± 0.8	$\mathbf{100.0 \pm} 0.0$	36.6 ± 4.6
SVHN	Threshold	66.4 ± 1.7	90.1 ± 0.3	85.9 ± 0.4	99.3 ± 0.0	42.8 ± 0.6
	DeepEnsemble	$\mathbf{67.2}$±NA	$91.0\pm$ NA	$86.6\pm$ NA	$\mathbf{99.4}\pm$ NA	$46.5\pm$ NA
	MC-dropout	65.3 ± 0.4	90.4 ± 0.6	85.5 ± 0.6	99.3 ± 0.0	45.0 ± 1.2
	PN	64.5 ± 0.7	84.0 ± 0.4	81.5 ± 0.2	98.2 ± 0.2	43.9 ± 1.1
	BBP	58.7 ± 2.1	91.8 ± 0.2	85.6 ± 0.7	99.1 ± 0.1	50.7 ± 0.9
	P-SGLD	64.2 ± 1.3	$\mathbf{93.0} \pm 0.4$	$\mathbf{87.1} \pm 0.4$	$\mathbf{99.4} \pm 0.1$	48.6 ± 1.8
	SDE-Net	65.5 ± 1.9	92.3 ± 0.5	86.8 ± 0.4	$\mathbf{99.4 \pm} 0.0$	$\mathbf{53.9} \pm 2.5$

4. Network Architecture

4.1. Classification Task

Downsampling layer:

```
        self.downsampling_layers = nn.Sequential(
        #change the in planes to 3 for SVHN
            nn.Conv2d(1, dim, 3, 1),
            norm(dim),
            nn.ReLU(inplace=True),
            nn.Conv2d(dim, dim, 4, 2, 1),
            norm(dim),
            nn.ReLU(inplace=True),
            nn.Conv2d(dim, dim, 4, 2, 1),
        )
```

Drift neural network:

```
class Drift(nn.Module):
    def __init__(self, dim):
        super(Drift, self).__init__()
        self.norm1 = norm(dim)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = ConcatConv2d(dim, dim, 3, 1, 1)
        self.norm2 = norm(dim)
        self.conv2 = ConcatConv2d(dim, dim, 3, 1, 1)
        self.norm3 = norm(dim)

    def forward(self, t, x):
        out = self.norm1(x)
        out = self.relu(out)
        out = self.conv1(t, out)
        out = self.norm2(out)
        out = self.relu(out)
        out = self.conv2(t, out)
        out = self.norm3(out)
        return out
```

Diffussion neural network for MNIST:

```
class Diffusion(nn.Module):
    def __init__(self, dim_in, dim_out):
        super(Diffusion, self).__init__()
        self.norm1 = norm(dim_in)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.norm2 = norm(dim_in)
        self.conv2 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.fc = nn.Sequential(norm(dim_out), nn.ReLU(inplace=True), nn.AdaptiveAvgPool2d
                                                    ((1, 1)), Flatten(), nn.Linear(
                                                    dim_out, 1), nn.Sigmoid())
    def forward(self, t, x):
        out = self.norm1(x)
        out = self.relu(out)
        out = self.conv1(t, out)
        out = self.norm2(out)
        out = self.relu(out)
        out = self.conv2(t, out)
        out = self.fc(out)
        return out
```

Diffusion network for SVHN:

```
class Diffusion(nn.Module):
    def __init__(self, dim_in, dim_out):
        super(Diffusion, self).__init__()
        self.norm1 = norm(dim_in)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.norm2 = norm(dim_in)
        self.conv2 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.norm3 = norm(dim_in)
        self.conv3 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.fc = nn.Sequential(norm(dim_out), nn.ReLU(inplace=True), nn.AdaptiveAvgPool2d
                                                            ((1, 1)), Flatten(), nn.Linear(
                                                            dim_out, 1), nn.Sigmoid())

    def forward(self, t, x):
        out = self.norm1(x)
        out = self.relu(out)
        out = self.conv1(t, out)
        out = self.norm2(out)
        out = self.relu(out)
        out = self.conv2(t, out)
        out = self.norm3(out)
        out = self.relu(out)
        out = self.conv3(t, out)
        out = self.fc(out)
        return out
```

ResNet block architecture:

```
class ResBlock(nn.Module):
    expansion = 1
    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(ResBlock, self).__init__()
        self.norm1 = norm(inplanes)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.norm2 = norm(planes)
        self.conv2 = conv3x3(planes, planes)

    def forward(self, x):
        shortcut = x

        out = self.relu(self.norm1(x))

        if self.downsample is not None:
            shortcut = self.downsample(out)

        out = self.conv1(out)
        out = self.norm2(out)
        out = self.relu(out)
        out = self.conv2(out)

        return out + shortcut
```

For BBP, we use an identical Residue block architecture and a fully factorised Gaussian approximate posterior on the weights.

4.2. Regression Task

The network architecture for DeepEnsemble, MC-dropout and p-SGLD:

```
class DNN(nn.Module):
    def __init__(self):
        super(DNN, self).__init__()
        self.fc1 = nn.Linear(90, 50)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(50, 50)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(50, 50)
        self.dropout3 = nn.Dropout(0.5)
        self.fc4 = nn.Linear(50, 50)
        self.dropout4 = nn.Dropout(0.5)
        self.fc5 = nn.Linear(50, 50)
        self.dropout5 = nn.Dropout(0.5)
        self.fc6 = nn.Linear(50, 2)

    def forward(self, x):
        x = self.dropout1(F.relu(self.fc1(x)))
        x = self.dropout2(F.relu(self.fc2(x)))
        x = self.dropout3(F.relu(self.fc3(x)))
        x = self.dropout4(F.relu(self.fc4(x)))
        x = self.dropout5(F.relu(self.fc5(x)))
        x = self.fc6(x)
        mean = x[:,0]
        sigma = F.softplus(x[:,1])+1e-3
        return mean, sigma
```

For BBP, we use an identical architecture with a fully factorised Gaussian approximate posterior on the weights.

For SDE-Net:

Drift neural network:

```
class Drift(nn.Module):
    def __init__(self):
        super(Drift, self).__init__()
        self.fc = nn.Linear(50, 50)
        self.relu = nn.ReLU(inplace=True)
    def forward(self, t, x):
        out = self.relu(self.fc(x))
        return out
```

Diffusion neural network:

```
class Diffusion(nn.Module):
    def __init__(self):
        super(Diffusion, self).__init__()
        self.relu = nn.ReLU(inplace=True)
        self.fc1 = nn.Linear(50, 100)
        self.fc2 = nn.Linear(100, 1)
    def forward(self, t, x):
        out = self.relu(self.fc1(x))
        out = self.fc2(out)
        out = F.sigmoid(out)
        return out
```

References

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. In *International Conference on Machine Learning*, pp. 1321–1330, 2017.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pp. 6402–6413, 2017.

Lalley, S. P. Stochastic differential equations. In *University of Chicago*, 2016.