Post-hoc loss-calibration for Bayesian neural networks (Supplementary material)

Meet P. Vadera [*1] **Soumya Ghosh**[2] **Kenney Ng**[2] **Benjamin M. Marlin**[1]

[1]College of Information and Computer Sciences, University of Massachusetts Amherst, Amherst, MA, USA
[2]IBM Research, Cambridge, MA, USA

A APPENDIX

A.1 POST-HOC CORRECTION ALGORITHM

We provide a high level overview of our method in Algorithm 1. Note that this algorithm assumes that we have access to a student model S that can approximate the posterior predictive distribution $p(y|\mathbf{x}, \mathcal{D})$.

Algorithm 1 Post-hoc loss correction algorithm

```
Require: Amortized posterior approximation $S(.|.,\omega)$, Calibration dataset $\mathcal{D}'$, minibatch size $B$, loss-calibrated model $q(.|.,\lambda)$, number of training iterations $T$, initialization $\lambda_0$, supremum of the loss function $\ell$, $M$.
 1: Initialize the loss calibrated model by setting its parameters to $\lambda_0$.
 2: for t $\in [1, \ldots, T]$ do
 3:     Draw a mini-batch $\mathcal{B} \subset \mathcal{D}'$ of size $B$.
 4:     for each data point $\mathbf{x}_b \in \mathcal{B}$ do
 5:         Compute $c_b = \arg\min_c \int_{y_b} \ell(c, y_b) q(y_b \mid \mathbf{x}_b, \lambda_t) dy_b$
 6:     end for
 7:     Define $\tilde{\mathcal{L}}(\lambda, \mathbf{c}; \mathcal{B}) = -\sum_{b=1}^{B} \mathbb{E}_{q(y_b|\mathbf{x}_b,\lambda)} \left[\frac{\ell(c_b, y_b)}{M}\right] - \mathrm{KL}(q(y_b|\mathbf{x}_b, \lambda)||S(y_b|\mathbf{x}_b, \omega)))$
 8:     Update $\lambda_{t+1} \leftarrow$ SGDUpdate$(\lambda_t, \nabla_\lambda \tilde{\mathcal{L}}(\lambda, \mathbf{c}, \mathcal{B}))$
 9: end for
10: return Optimized parameters of the loss calibrated model, $\lambda_T$.
```

[*]Work partially done during internship at IBM Research.

A.2 DERIVATION OF THE POST-HOC CORRECTION OBJECTIVE

We begin our derivation with the definition of log conditional gain as follows:

$$\begin{aligned}
&\log \mathcal{G}(\mathbf{h} = \mathbf{c} \mid \mathcal{D}'; \lambda) \\
&= \sum_{n=1}^{N} \log \int_y \Bigg(u(h = c_n, y_n = y) q(y_n = y|\mathbf{x_n}, \lambda) \\
&\qquad\qquad \times \frac{p(y_n = y|\mathbf{x_n}, \mathcal{D}, \theta^0)}{q(y_n = y|\mathbf{x_n}, \lambda)} dy \Bigg) \\
&= \sum_{n=1}^{N} \log \int_y \Bigg(q(y_n = y|\mathbf{x_n}, \lambda) \\
&\qquad\qquad \times \left(\frac{p(y_n = y|\mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)}{q(y_n = y|\mathbf{x_n}, \lambda)} \right) \Bigg) dy \\
&= \sum_{n=1}^{N} \log \mathbb{E}_{q(y_n = y|\mathbf{x_n}, \lambda)} \left[\frac{p(y_n = y|\mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)}{q(y_n = y|\mathbf{x_n}, \lambda)} \right]
\end{aligned}$$

Now, using Jensen's inequality, we obtain,

$$\begin{aligned}
\log \mathcal{G}(\mathbf{h} = \mathbf{c}|\mathbf{X}) &\geq \sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n}, \lambda)} \left[\log \left(\frac{p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) u(c_n, y_n)}{q(y)_n|\mathbf{x_n}, \lambda)} \right) \right] \\
&= \sum_{n=1}^{N} \mathbb{E}_{q(y_n|\mathbf{x_n}, \lambda)} \left[\log u(c_n, y_n) \right] \\
&\quad - \mathrm{KL}(q(y_n|\mathbf{x_n}, \lambda) || p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0)) \\
&:= \mathcal{U}(\lambda, \mathbf{c}; \mathcal{D}')
\end{aligned} \tag{1}$$

A.3 THE VARIATIONAL GAP

Consider a single data instance $\mathbf{x}_n \in \mathcal{D}'$. The gap between the log conditional gain and the lower bound then is,

$$\begin{aligned}
&\log \mathcal{G}(h = c_n \mid \mathcal{D}'; \lambda) - \mathcal{U}(\lambda, c_n, \mathcal{D}') = \log \int u(h = c_n, y_n = y) p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) dy - \mathcal{U}(\lambda, c_n, \mathcal{D}') \\
&= \log \int u(h = c_n, y_n = y) p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) dy - \int q(y_n = y|\mathbf{x}_n, \lambda) \log u(h = c_n, y_n = y) dy \\
&+ \int q(y_n = y|\mathbf{x}_n, \lambda) \log q(y_n = y|\mathbf{x}_n, \lambda) dy - \int q(y_n = y|\mathbf{x}_n, \lambda) \log p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) \\
&= \log \int u(h = c_n, y_n = y) p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) dy + \int q(y_n = y|\mathbf{x}_n, \lambda) \log \frac{q(y_n = y|\mathbf{x}_n, \lambda)}{p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)} dy \\
&= \log Z_n + \int q(y_n = y|\mathbf{x}_n, \lambda) \log \frac{q(y_n = y|\mathbf{x}_n, \lambda)}{p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)} dy \\
&= \int q(y_n = y|\mathbf{x}_n, \lambda) \log Z_n dy + \int q(y_n = y|\mathbf{x}_n, \lambda) \log \frac{q(y_n = y|\mathbf{x}_n, \lambda)}{p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)} dy \\
&= \int q(y_n = y|\mathbf{x}_n, \lambda) \log \frac{q(y_n = y|\mathbf{x}_n, \lambda) Z_n}{p(y_n|\mathbf{x_n}, \mathcal{D}, \theta^0) u(h = c_n, y_n = y)} dy \\
&= \mathrm{KL}\big[q(y_n|\mathbf{x}_n, \lambda) || \frac{p(y_n|\mathbf{x}_n, \mathcal{D}, \theta^0) u(h = c_n, y_n)}{Z_n}\big].
\end{aligned} \tag{2}$$

Summing over all data points in $\mathcal{D}'$ gives us Eq. (7).

A.4 OVERVIEW OF STOCHASTIC GRADIENT HAMILTONIAN MONTE CARLO (SGHMC)

Since we use SGHMC often as a part of this paper, we provide a brief overview of SGHMC in this appendix. SGHMC is a SGMCMC algorithm, which is a class of MCMC algorithms that function based on mini-batch gradients. This is particularly helpful for deep learning as traditional models and data sets are too large to compute the gradients of the likelihood on entire data, which can lead to a bottleneck in computation. SGHMC algorithm involves computing the $\tilde{U}(\theta)$ on a minibatch $\mathcal{B}$ as shown in the first equation, followed by running the next two update equations:

$$\tilde{U}(\theta) = -\log(p(\mathcal{B}|\theta)) - \log(p(\theta|\theta^0)) \tag{3}$$

$$\theta_k = \theta_{k-1} + v_{k-1} \tag{4}$$

$$v_k = v_{k-1} - \alpha_k \nabla\tilde{U} - \eta v_{k-1} + \sqrt{2(\eta - \hat{\gamma})\alpha_k}\epsilon_k \tag{5}$$

In the above equations k denotes the iteration, v_k denotes the momentum term, $(1-\eta)$ denotes the momentum factor, η_k is drawn from an identity gaussian distribution, $\nabla\tilde{U}$ denotes the gradient approximation obtained using a minibatch, $\hat{\gamma}$ and α_k denotes the instantaneous step size. The process highlighted in the above equations is run iteratively to draw new samples from the posterior. For practical purposes, most implementations set $\hat{\gamma}$ to 0 [Zhang et al., 2020, Vadera et al., 2020]. Interestingly, SGLD can be derived from SGHMC by setting the momentum factor to 0. Our experiments use the SGHMC implementation provided by Vadera et al. [2020].

A.5 ADDITIONAL EXPERIMENTAL DETAILS AND RESUTS: SYNTHETIC DATA EXPERIMENTS

Additional experimental Details: For the SGHMC implementation in this experiment, we use a fixed learning rate of 0.1, a momentum of 0.5 and prior precision of 1.0. We run a burn-in phase of 300 iterations, and collect total 100 parameter samples with a thinning interval of 50 iterations. After this, we obtain the monte carlo approximation to the posterior predictive distribution $p(y|\mathbf{x}, \mathcal{D})$ on the additional training data points ($\mathcal{D}'$). For doing our post hoc correction, we use an MLP with 50 hidden units (as with the original model) and optimize the objective shown in Eq. (12) using Adam optimizer with a learning rate of 0.1 for 500 training iterations.

For BBVI implementation in this experiment, we set an identity gaussian distribution as our prior and maximize the evidence lower bound (ELBO) of our MLP-BNN using Adam optimizer with a learning rate of 0.01 over 5000 iterations. We use the local-reparameterization trick Kingma et al. [2015] to produce low variance stochastic gradients. Next, we again collect a total of 100 parameter samples and compute the Monte Carlo approximation of the posterior predictive distribution $p(y|\mathbf{x}, \mathcal{D})$ on the additional training data points ($\mathcal{D}'$). For doing our post hoc correction, we use an MLP with 50 hidden units (as with the original model) as our $q(.)$ model, and optimize the objective shown in Eq. (12) using Adam optimizer with a learning rate of 0.1 for 500 training iterations.

For KFAC-Laplace we perform a Laplace approximation about the maximum-a-posteriori (MAP) solution. As in standard Laplace approximation, the approximate posterior is represented by a Gaussian centered at the MAP solution with its covariance set to the inverse of the Hessian. We find the MAP solution by using Adam with a learning rate of 0.01 to maximize the negative log posterior. Following Ritter et al. [2018] we use a block-diagonal, Kronecker-factored Hessian.

Finally, note that there are 100 training points in the original training set ($\mathcal{D}$), 500 additional unlabeled data points for our posterior correction ($\mathcal{D}'$), and 100 held out data points for testing from the same data distribution as $\mathcal{D}$.

Additional experimental Details: The full panel of results is given in Table 1.

Table 1: **Results on synthetic data**. Test decision costs with and without post-hoc correction over 10 replicates. Post-hoc correction consistently provides lower cost decisions. Results presented as mean ± std. dev.

	W/O post-hoc correction	W/ post-hoc correction (ours)	Avg. paired diff.	Std. (avg. paired diff.)
VI	0.019 ± 0.011	0.016 ± 0.010	0.00146	0.001
SGHMC	0.018 ± 0.008	0.017 ± 0.009	0.001	0.006
KFAC-Laplace	0.021 ± 0.007	0.018 ± 0.008	0.002	0.005

A.6 ADDITIONAL EXPERIMENTAL DETAILS AND RESULTS: SELECTIVE DECISION MAKING

Additional experimental Details: In this experiment, we employ two methods for approximating the posterior predictive distribution $p(y|\mathbf{x}, \mathcal{D}, \theta^0)$: using a student model and pre-computing the posterior predictive distribution using the samples from SGHMC. For both cases, we generate the additional unlabeled training data $\mathcal{D}'$ by applying the same random transformation on original CIFAR10 data set, and generate 10 copies for every example by randomly rotating the image as described earlier. In the case where we use the student model, we distill the posterior predictive distribution using the approach of Balan et al. [2015] and use the same $\mathcal{D}'$ for the distillation. Note that the approach by Balan et al. [2015] allows us to interleave the sampling from $p(\theta|\mathcal{D}, \theta')$ and distilling to a student model using an online approach.

Finally, once we obtain some form of approximation for the posterior predictive distribution, we optimize either Eq. (12) or Eq. (13) depending on whether we have used the student or not. We use the same ResNet18 architecture for $q(.)$ model. For the SGHMC chains, we use a momentum of 0.7, a fixed learning rate of 10^{-3}, and a prior precision of 5, and 1000 burn-in iterations (each iteration is a gradient step after a mini-batch). We run the SGMCMC sampling-distillation algorithm from Balan et al. [2015] for a total of 100 epochs, and collect a total of 30 samples at the end of each of the last 30 epochs. The student model has the same ResNet18 architecture, and is trained using SGD with a one-cycle learning rate schedule [Smith, 2017]. The peak learning rate is 0.1 and the weight decay factor is 5×10^{-4}. We warm start our $q(.)$ model with the student and train it for 100 epochs using an SGD optimizer with a learning rate of 10^{-3}. For the case where we don't use a student model, we train our $q(.)$ model using SGD with a cyclic learning rate schedule as it helps accelerate the training. The peak learning rate is 0.1 and the weight decay factor is . We use the equivalent number of training iterations as 100 epochs on the original data set $\mathcal{D}$. For sampling a mini-batch from $\mathcal{D}'$, we use sampling without replacement at every iteration. The mini-batch size for the entire experiment is set to 64. Also note that the SGHMC chain begins with a pretrained maximum-a-posteriori (MAP) solution. This pre-trained solution is obtained using an SGD optimizer with a one-cycle learning rate scheduler, with a peak learning rate of 0.05, with the same prior precision, and is trained for 100 epochs using $\mathcal{D}$.

Additional results: In Figure 2 we provide the NLL (↓) results (on unrefered data points) from the selective decision making experiment.

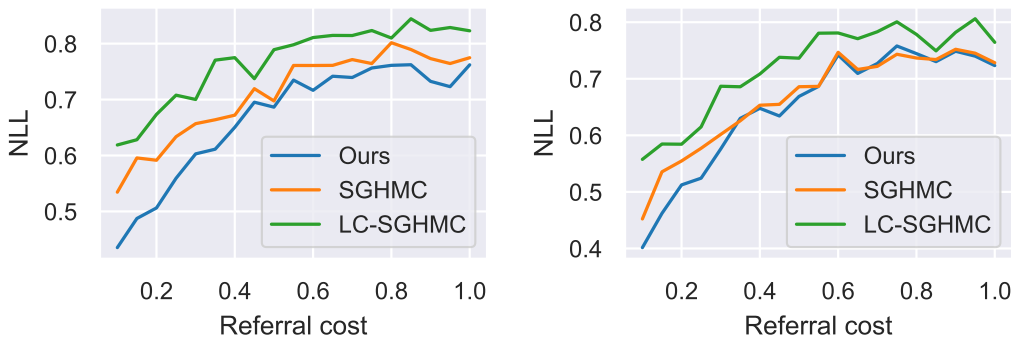

Figure 1: **Selective classification**. NLL comparison on unreferred data points for different levels of label corruption for CIFAR10 data set using ResNet18 model. Top plot represent results from the experiment not using the student S to approximate the posterior predictive distribution while the bottom plot represents results from the experiment using the student S to approximate the posterior predictive distribution.

A.7 ADDITIONAL EXPERIMENTAL DETAILS AND RESULTS: DECISION MAKING UNDER POOR DATA QUALITY

Additional experimental Details: In this experiment, we use a ResNet18 model with CIFAR10 data set and a MLP with a single hidden layer of 200 units (MLP200) for MNIST. We generate $\mathcal{D}'$ online, by adding random pixel noise from a $\mathcal{N}(0, 0.05)$ distribution. Since we use student model, we follow the same algorithm and subsequent post-hoc correction as described in the previous subsection of the appendix. Student model S and our $q(.)$ model follow the same architecture as of the original model.

For ResNet18-CIFAR10 combination, we use SGLD with a fixed learning rate of 2×10^{-5} and a prior precision of 50. The burn-in iterations is set to 10,000. We run the distillation algorithm to train the student model for 50 epochs, and collect 30 samples from the original model (also the teacher model here) from the end of the last 30 epochs. To train student model, we use SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.05, with a momentum of 0.9

and a weight decay factor of 10^{-4}. Once we obtain the student, we train our $q(.)$ model using the objective in Eq. (12). For training the $q(.)$ model, we use SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.05, with a momentum of 0.9. Note that for running SGHMC on ResNet18-CIFAR10 combination, we start with a pre-trained solution. This pre-trained MAP solution is obtained by running our original teacher model using the same data set $\mathcal{D}$. For the pre-training step, we use SGD with a cosine annealing learning with warm restarts. The initial learning rate in the schedule is 0.05, with a momentum of 0.9 and a weight decay factor of 5×10^{-4}. Each annealing phase is of 20 epochs, and we perform total 5 such phases.

For MNIST-MLP200 combination, we use SGLD (obtained by setting $\alpha = 1$ in SGHMC) with a fixed learning rate of 10^{-4} and a prior precision of 6. The burn-in iterations is set to 10,000. We run the distillation algorithm to train the student model for 100 epochs, and collect 30 samples from the original model (also the teacher model here) from the end of the last 30 epochs. To train the student model, we use SGD with a fixed learning rate of 10^{-3}, with a momentum of 0.9 and a weight decay factor of 10^{-4}. Once we obtain the student, we train our $q(.)$ model using the objective in Eq. (12). For training the $q(.)$ model,we use SGD with a fixed learning rate of 10^{-3}, with a momentum of 0.9. Note that for running SGLD on MNIST-MLP200 combination, we start with a pre-trained MAP solution as well. For the pre-training step, we use SGD with a cosine annealing learning with warm restarts. The initial learning rate in the schedule is 0.01, with a momentum of 0.9 and a weight decay factor of 1×10^{-4}. Each annealing phase is of 20 epochs, and we perform total 5 such phases.

The decision cost function for CIFAR10 is defined as shown below:

$$\ell(c, y) = \begin{cases} 0, & \text{for } y = c, \\ 0.7, & \text{for } y \neq c, \text{ and c} \in \{\text{automobile, truck}\} \\ 1.0, & \text{otherwise} \end{cases} \tag{6}$$

The decision cost function for MNIST is defined as shown below:

$$\ell(c, y) = \begin{cases} 0, & \text{for } y = c, \\ 0.7, & \text{for } y \neq c, \text{ and c} \in \{3, 8\} \\ 1.0, & \text{otherwise} \end{cases} \tag{7}$$

For the CW-SGD baseline, we assign a loss weight of 1.4 to the instance if the ground truth is either automobile or truck for CIFAR10 and if the ground truth either 3 or 8 for MNIST. The reason for this is that we want to our system to assign higher importance to these classes, and thus a mistake of making an incorrect decision with these ground truth classes should incur higher loss. Similarly, the decision cost function highlights the same fact as it places a lesser penalty if our decision system predicts these important classes.

Additional results: In Figure 2 we provide the NLL (↓) comparison from the experiment on decision making under poor data quality.

Table 2: **Semantic scene segmentation.** Decision cost matrix for the semantic scene segmentation experiment.

		Decision											
	Cost	Sk.	Bu.	Po.	Ro.	Pa.	Tr.	Si.	Fe.	Ca.	Pe.	Cy.	Un.
Ground Truth/Prediction	Sky	0.	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.6
	Building	0.8	0.	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.6
	Pole	0.8	0.8	0.	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.6
	Road	0.8	0.8	0.6	0.	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.6
	Pavement	0.8	0.8	0.6	0.6	0.	0.6	0.6	0.6	0.4	0.4	0.4	0.6
	Tree	0.8	0.8	0.6	0.6	0.6	0.	0.6	0.6	0.4	0.4	0.4	0.6
	Sign	0.8	0.8	0.6	0.6	0.6	0.6	0.	0.6	0.4	0.4	0.4	0.6
	Fence	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.	0.4	0.4	0.4	0.6
	Car	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.	0.4	0.4	0.6
	Pedestrian	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.	0.2	0.6
	Cyclist	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.2	0.	0.6
	Unlabelled	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.4	0.

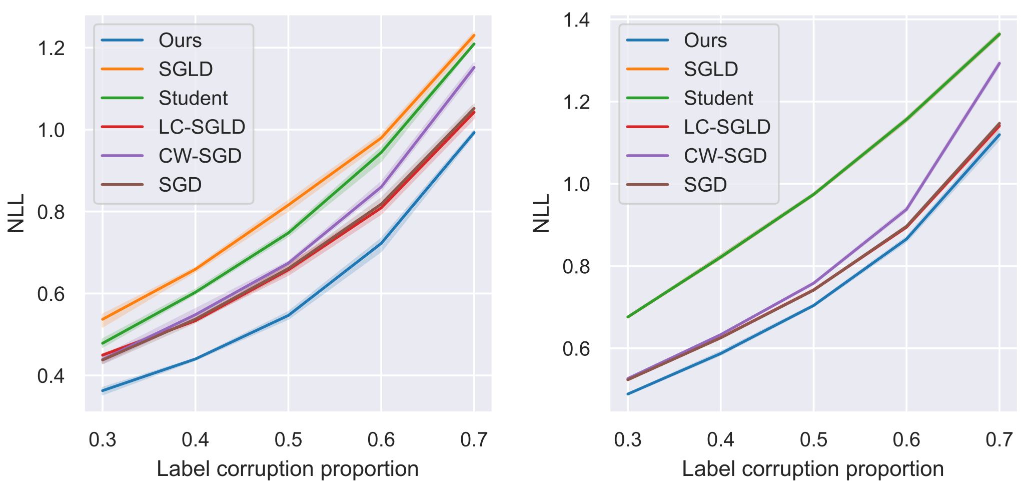

Figure 2: **Decision-making under label corruption.** (Top) Performance comparison for different levels of label corruption for CIFAR10 data set using ResNet18 model and (Bottom) MNIST data set using an MLP with a single hidden layer of 200 hidden units. The results are shown as mean $\pm$ std. dev. over 5 trials.

A.8 ADDITIONAL EXPERIMENTAL DETAILS: SEMANTIC SCENE SEGMENTATION

For the SGHMC implementation in this experiment, we use an initial learning rate of 0.01, a momentum of 0.5 and effective weight decay of 5×10^{-4}. We also use a cosine annealing learning rate scheduler which decays the learning rate to 0. We run a burn-in phase of 1000 iterations, and collect total 30 parameter samples with a thinning interval of 500 iterations. We run the distillation algorithm alongside the sampling to train student model for a total of 15000 iterations after the burn-in. To train student model, we use SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.15, with a momentum of 0.9 and a weight decay factor of 10^{-4}. Once we obtain the student, we train our $q(.)$ model using the objective in Eq. (12). For training the $q(.)$ model,we use SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.05, with a momentum of 0.9. Note that for running SGHMC we start with a pre-trained solution. The pre-trained solution used in this experiment is the CW-SGD solution. More details on CW-SGD solution is given later in this section. Once we obtain the student model S, we train the $q(.)$ model, using SGD with a cyclic learning rate schedule. The maximum learning rate in the schedule is 0.05, with a momentum of 0.9 for a total of 15000 iterations.

For the CW-SGD baseline, we use a cyclic learning rate schedule and train the Segnet model for 15000 iterations with a maximum learning rate of 0.05, weight decay of 10^{-4} and momentum of 0.9. The classes of importance here are the car, cyclist and pedestrian, and hence the loss weights for these classes are set to 1.4 while rest of the loss weights are set to 1.

For the loss calibrated MC dropout baseline, we use the pretrained SGD solution, and fine-tune it with Adam optimizer using a learning rate of 10^{-4} for 1000 training iterations. The dropout strength is 0.3 and is added in the decoding phase of the model. For computing the posterior predictive distribution at train, we use 5 forward passes, and during testing, we use 30 forward passes. Additional details about this algorithm can be found in Cobb et al. [2018]

The decision cost matrix for this experiment is shown in Table 2. This decision cost matrix puts less penalty on decisions that by mistake choose to classify a pixel belong to one of the high importance classes.

References

Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *NeurIPS*, 2015.

Adam D. Cobb, Stephen J. Roberts, and Yarin Gal. Loss-calibrated approximate inference in bayesian neural networks. In *ICML theory of deep learning workshop*, 2018.

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *NeurIPS*, 2015.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *ICLR*, volume 6, 2018.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

Meet P. Vadera, Adam D. Cobb, Brian Jalaian, and Benjamin M. Marlin. Ursabench: Comprehensive benchmarking of approximate bayesian inference methods for deep neural networks. *arXiv preprint arXiv:2007.04466*, 2020. URL `https://github.com/reml-lab/URSABench`.

Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient mcmc for bayesian deep learning. In *ICLR*, 2020.