Practical Deep Learning with Bayesian Principles

Kazuki Osawa[1], Siddharth Swaroop[2], Anirudh Jain[3], Runa Eschenhagen[4], Richard E. Turner[2], Rio Yokota[1], Mohammad Emtiyaz Khan[5]

[1]Tokyo Institute of Technology, Japan [2]University of Cambridge, UK [3]Indian Institute of Technology (ISM), India
[4]University of Osnabrück, Germany [5]Center for Advanced Intelligence Project, RIKEN, Japan

Introduction

Motivation:

- **Benefits of Bayesian principles** in learning:
  - represent *uncertainty* using the posterior distribution
  - enable sequential learning using Bayes' rule
  - reduce overfitting with Bayesian model averaging
- Bayesian methods are *impractical* in deep learning and rarely match the performance of standard methods (e.g., Adam optimiser).

Contributions:

- Demonstrate practical training of deep networks with **natural-gradient variational inference (NGVI)** with existing deep learning techniques.
  - predictive probabilities are well-calibrated
  - uncertainties on out-of-distribution data are improved
  - continual-learning performance is boosted
- Achieve similar convergence as the Adam optimiser **on ImageNet for the first time** while **preserving the benefits of Bayesian principles**.

Why Bayes?

VOGN [2], an Adam-like optimiser based on the Bayesian principles, can obtain uncertainty in Deep Learning.

Uncertainty in 2D binary-classification

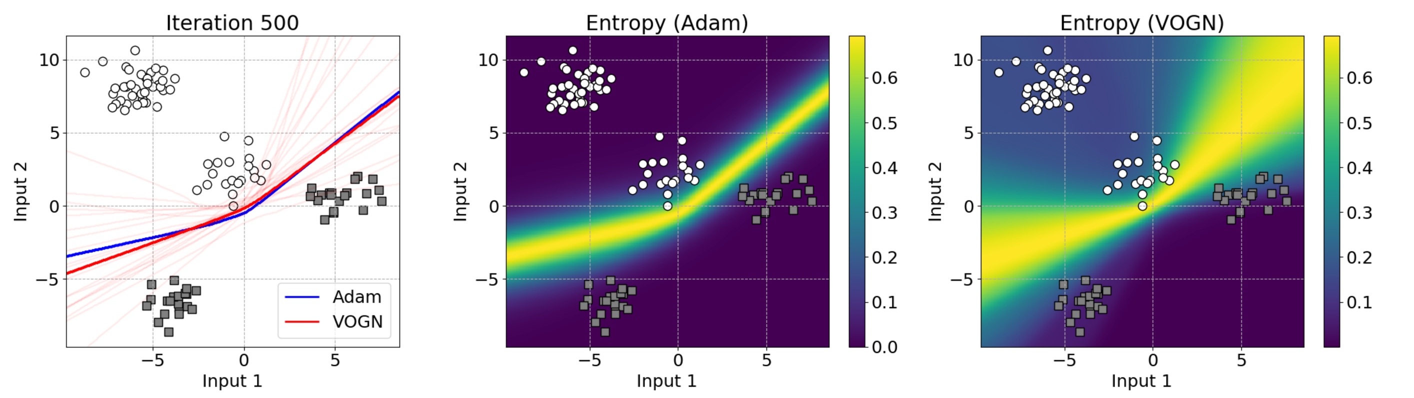


Figure: Decision boundary and entropy plots by MLPs trained with Adam and VOGN. VOGN optimises the posterior distribution of each weight (i.e., mean and variance of the Gaussian). A model with the mean weights draws the red boundary, and models with the MC samples from the posterior distribution draw light red boundaries. VOGN converges to a similar solution as Adam while keeping uncertainty in its predictions.

Uncertainty in Image Segmentation

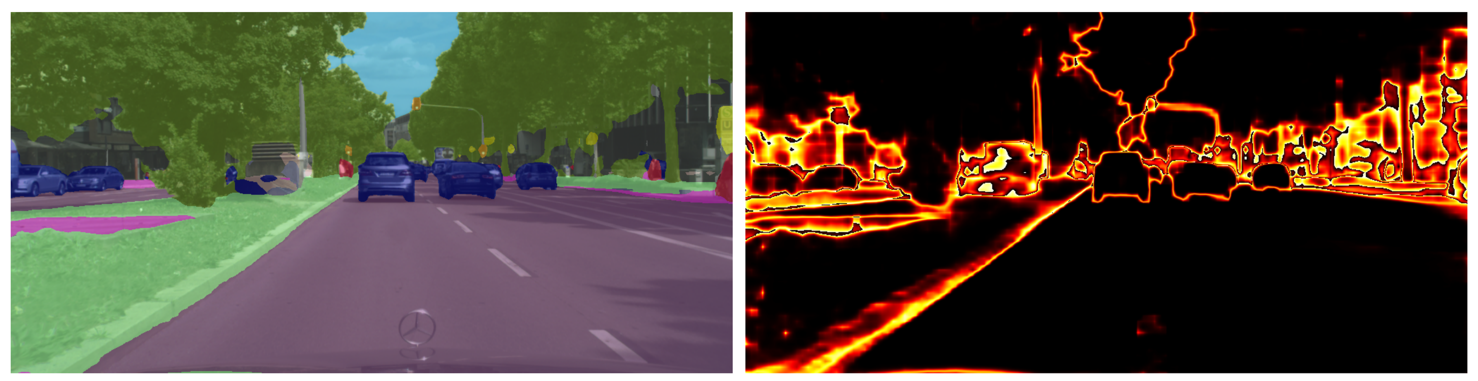


Figure: Segmentation (left) and its predictive entropy (right) by VOGN on Cityscapes dataset (the figures were created by Roman Bachmann, EPFL.)

How to apply the Bayesian method (VOGN) to my project?

A PyTorch implementation is available as a plug-and-play optimiser.

```
import torch
+import torchsso

train_loader = torch.utils.data.DataLoader(train_dataset)
model = MLP()

-optimizer = torch.optim.Adam(model.parameters())
+optimizer = torchsso.optim.VOGN(model, dataset_size=len(train_loader.dataset))
```

https://github.com/team-approx-bayes/dl-with-bayes

Natural Gradient Variational Inference (NGVI)

Objective for *Standard Deep Learning*:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \bar{\ell}(\mathbf{w}) + \delta \mathbf{w}^\top \mathbf{w}, \text{ where } \bar{\ell}(\mathbf{w}) := \frac{1}{N}\sum_i \ell(\mathbf{y}_i, \mathbf{f}_w(\mathbf{x}_i)),$$

Stochastic Gradient optimisers (e.g., SGD, RMSprop, Adam)

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha_t \frac{\hat{\mathbf{g}}(\mathbf{w}_t) + \delta \mathbf{w}_t}{\sqrt{\mathbf{s}_{t+1}} + \epsilon}, \qquad \mathbf{s}_{t+1} \leftarrow (1-\beta_t)\mathbf{s}_t + \beta_t \left(\hat{\mathbf{g}}(\mathbf{w}_t) + \delta \mathbf{w}_t\right)^2,$$

$\alpha_t > 0, 0 < \beta_t < 1$: learning rates, $\hat{\mathbf{g}}(\mathbf{w}) := \frac{1}{M}\sum_{i\in\mathcal{M}_t} \nabla_w \ell(\mathbf{y}_i, \mathbf{f}_w(\mathbf{x}_i))$ with a minibatch $\mathcal{M}_t$ of M data.

Objective for *Bayesian Deep Learning (Gaussian Variational Inference)*:

$$\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^* = \arg\max_{\boldsymbol{\mu}, \boldsymbol{\Sigma}} -N\mathbb{E}_q\left[\bar{\ell}(\mathbf{w})\right] - \tau \mathbb{D}_{KL}[q(\mathbf{w}) \,\|\, p(\mathbf{w})].$$

$p(\mathbf{w})$: prior distribution, $q(\mathbf{w}) := \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \approx p(\mathbf{w}|\mathcal{D})$: Gaussian approximate posterior

Natural Gradient Variational Inference (NGVI) update takes a simple form when estimating exponential-family approximations [1].

$$\boldsymbol{\lambda}_{t+1} = (1-\tau\rho)\boldsymbol{\lambda}_t - \rho \nabla_{\boldsymbol{\mu}} \mathbb{E}_q\left[\bar{\ell}(\mathbf{w}) + \tfrac{1}{2}\tau\delta \mathbf{w}^\top \mathbf{w}\right].$$

$p(\mathbf{w}) := \mathcal{N}(\mathbf{w}|0, \mathbf{I}/\delta)$, $\boldsymbol{\lambda}$: the natural-parameter

Variational Online Gauss-Newton (VOGN) [2]

$$\boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t - \alpha_t \frac{\hat{\mathbf{g}}(\mathbf{w}_t) + \tilde{\delta}\boldsymbol{\mu}_t}{\mathbf{s}_{t+1} + \tilde{\delta}}, \qquad \mathbf{s}_{t+1} \leftarrow (1-\tau\beta_t)\mathbf{s}_t + \beta_t \frac{1}{M}\sum_{i\in\mathcal{M}_t} (\mathbf{g}_i(\mathbf{w}_t))^2,$$

$\mathbf{g}_i(\mathbf{w}_t) := \nabla_w \ell(y_i, f_{w_t}(\mathbf{x}_i))$, $\mathbf{w}_t \sim \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, $\tilde{\delta} := \tau\delta/N$, $\boldsymbol{\Sigma}_t := \text{diag}(1/(N(\mathbf{s}_t + \tilde{\delta})))$.

NGVI + Deep Learning Techniques

Since VOGN takes a similar form to common optimisers, we can easily borrow existing deep-learning techniques to improve performance.

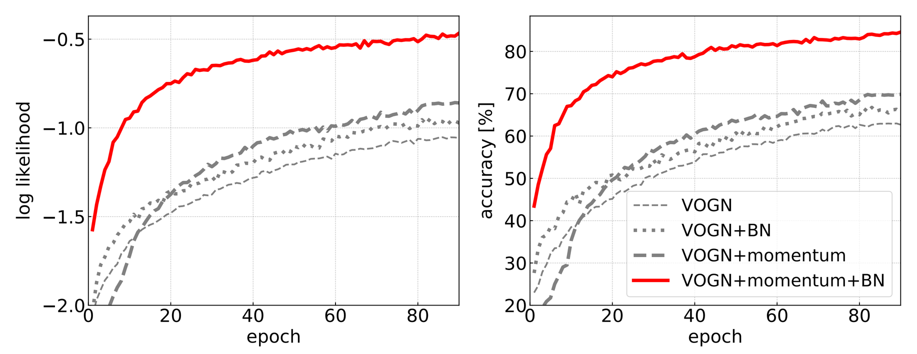


Figure: Effect of momentum and batch normalisation for training ResNet-18 on CIFAR-10.

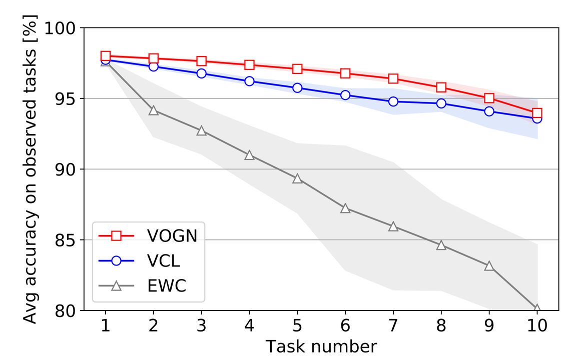


Figure: Continual learning task on Permuted MNIST.

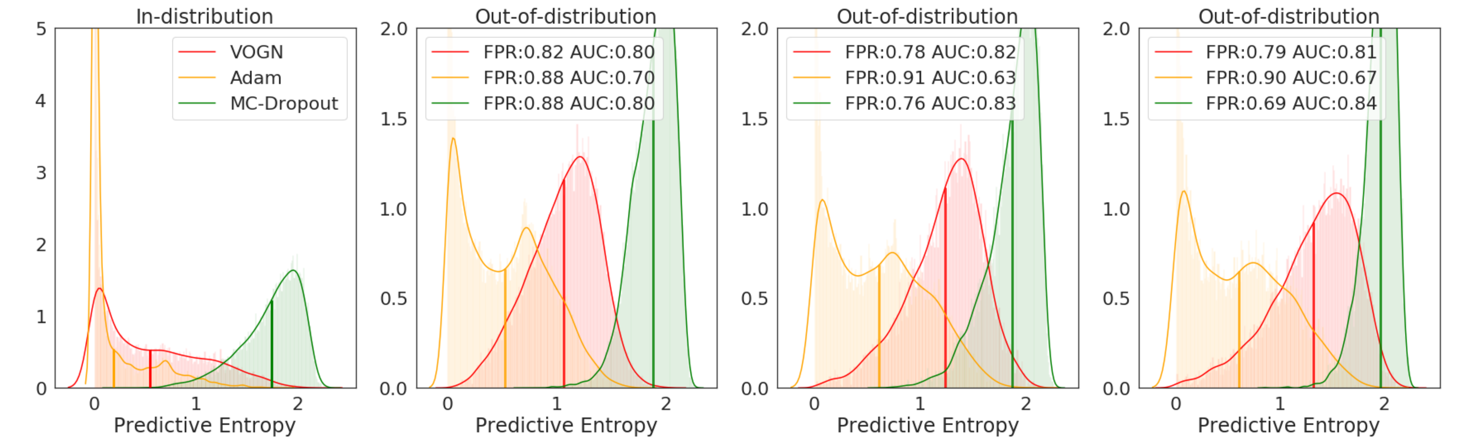


Figure: Histograms of predictive entropy for out-of-distribution tests for ResNet-18 trained on CIFAR-10. Left: in-distribution dataset (CIFAR-10). Right: out-of-distribution data: SVHN, LSUN (crop), LSUN (resize) (right). FPR at 95% TPR metric (lower is better) and the AUROC metric (higher is better). VOGN's predictive entropy is generally low for in-distribution and high for out-of-distribution data, but this is not the case for other methods.

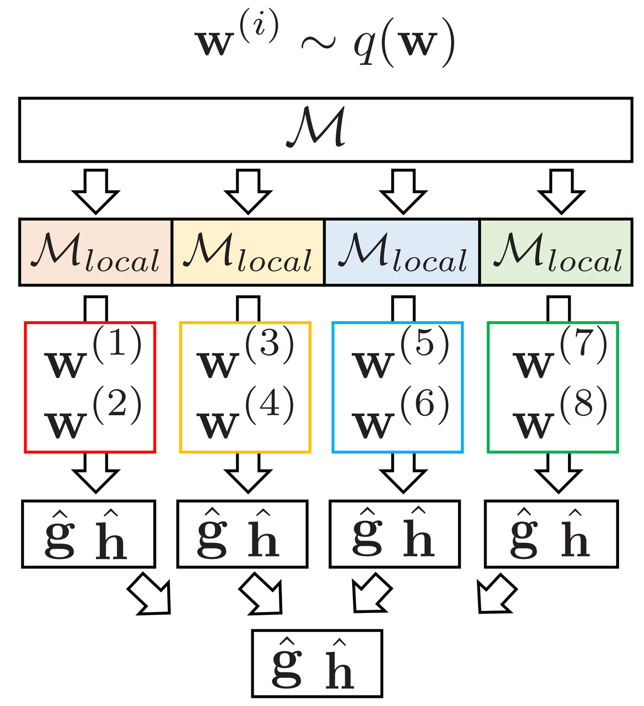
NGVI + Distributed Deep Learning

We employ a combination of the following two parallelism techniques with different Monte-Carlo (MC) samples for different inputs.

- **Data parallelism**: different GPUs process different inputs (local minibatches) *for accelerating the training* (more data in a step).
- **MC sample parallelism**: different GPUs use different MC samples from the posterior *for stabilising the training* (more samples in a step).

Algorithm Distributed VOGN

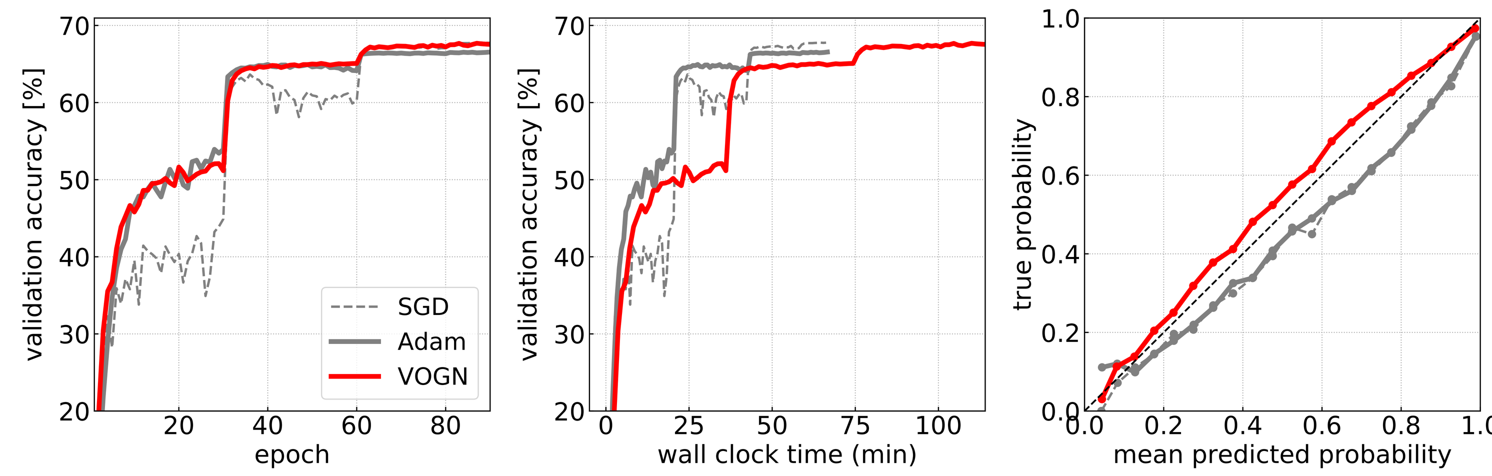
- **repeat**
  - Sample a minibatch $\mathcal{M}$ of size M.
  - Split $\mathcal{M}$ into each GPU (local minibatch $\mathcal{M}_{local}$).
  - **for** each GPU in parallel **do**
    - **for** $k = 1, 2, \ldots, K$ (# MC samples) **do**
      - Sample weight $\mathbf{w}^{(k)} \sim q(\mathbf{w})$.
      - Compute gradient $\mathbf{g}_i^{(k)} \forall i \in \mathcal{M}_{local}$
      - $\hat{\mathbf{g}}_k \leftarrow \frac{1}{M}\sum_{i\in\mathcal{M}_{local}} \mathbf{g}_i^{(k)}$ and $\hat{\mathbf{h}}_k \leftarrow \frac{1}{M}\sum_{i\in\mathcal{M}_{local}} (\mathbf{g}_i^{(k)})^2$.
    - **end for**
    - $\hat{\mathbf{g}} \leftarrow \frac{1}{K}\sum_{k=1}^K \hat{\mathbf{g}}_k$ and $\hat{\mathbf{h}} \leftarrow \frac{1}{K}\sum_{k=1}^K \hat{\mathbf{h}}_k$.
  - **end for**
  - AllReduce (aggregate) $\hat{\mathbf{g}}, \hat{\mathbf{h}}$ among all GPUs.
  - $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (\hat{\mathbf{g}} + \tilde{\delta}\boldsymbol{\mu})$ and $\mathbf{s} \leftarrow (1-\tau\beta_2)\mathbf{s} + \beta_2 \hat{\mathbf{h}}$.
  - $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha \mathbf{m}/(\mathbf{s} + \tilde{\delta})$.
- **until** stopping criterion is met =0



Bayes (NGVI) for ImageNet classification

Training ResNet-18 on ImageNet (1000 class) with 128 GPUs

VOGN has similar convergence behaviour as Adam and gives calibrated predictive probabilities (the diagonal represents perfect calibration).



Dataset/ Architecture	Optimiser	Train/Validation Accuracy (%)	Validation NLL	Epochs	Time/ epoch (s)	ECE	AUROC
	SGD	82.63 / **67.79**	**1.38**	90	44.13	0.067	0.856
	Adam	80.96 / 66.39	1.44	90	44.40	0.064	0.855
ImageNet/	MC-dropout	72.96 / 65.64	1.43	90	45.86	**0.012**	0.856
ResNet-18	**VOGN**	73.87 / **67.38**	**1.37**	90	76.04	0.029	0.854
	K-FAC	83.73 / 66.58	1.493	60	133.69	0.158	0.842
	Noisy K-FAC	72.28 / 66.44	1.44	60	179.27	0.080	0.852

Table: **DA**: Data Augmentation, **NLL**: Negative Log Likelihood, **ECE**: Expected Calibration Error, **AUROC**: Area Under ROC curve. Out of the 15 metrics (NLL, ECE, and AUROC on 5 dataset/architecture combinations), VOGN performs the best or tied best on 10, and is second-best on the other 5.

References

[1] M. E. Khan and D. Nielsen. Fast yet simple natural-gradient descent for variational inference in complex models. *CoRR*, abs/1807.04489, 2018.

[2] M. E. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In *Proceedings of 35 ICML*, pages 2611–2620, 2018.

Contact: oosawa.k.ad@m.titech.ac.jp