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:

- Bayesian methods promise to fix many shortcomings of deep learning, but they are impractical and rarely match the performance of standard methods, let alone improve them.

Contributions:

- We demonstrate practical training of deep networks with **natural-gradient variational inference** with existing deep learning techniques.
- We achieve **similar convergence as the Adam optimiser** on ImageNet.
- This work enables practical deep learning while preserving benefits of Bayesian principles : predictive probabilities are well-calibrated, uncertainties on out- of-distribution data are improved, and continual-learning performance is boosted.

Natural Gradient Variational Inference (NGVI)

In a supervised leraning task, with a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, we minimise a loss of the following form w.r.t. the weights $\mathbf{w}$:

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

$\mathbf{f}_w(\mathbf{x}) \in \mathbb{R}^K$:the DNN outputs with weights $\mathbf{w}$, $\ell(\mathbf{y}, \mathbf{f})$: a differentiable loss function between an output $\mathbf{y}$ and the function $\mathbf{f}$, $\delta > 0$: the L_2 regulariser.

Common optimisers (e.g., SGD, RMSprop, Adam) update $\mathbf{w}$ by:

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

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

In contrast, the Bayesian approach aims to obtain the posterior distribution of $\mathbf{w}$ using Bayes' rule: $p(\mathbf{w}|\mathcal{D}) = \exp\left(-N\bar{\ell}(\mathbf{w})/\tau\right) p(\mathbf{w})/p(\mathcal{D})$.

Gaussian Variational Inference computes a Gaussian approximation $q(\mathbf{w}) := \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \approx p(\mathbf{w}|\mathcal{D})$ by maximizing the ELBO w.r.t. $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:

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

Natural Gradient Variational Inference (NGVI) update takes a simple form when estimating exponential-family approximations [1]. When $p(\mathbf{w}) := \mathcal{N}(\mathbf{w}|0, \mathbf{I}/\delta)$, the update of the natural-parameter $\boldsymbol{\lambda}$ is performed by using the stochastic gradient of the *expected regularised-loss*:

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

Variational Online Gauss-Newton (VOGN) [2] estimates a Gaussian with mean $\boldsymbol{\mu}_t$ and a diagonal covariance matrix $\boldsymbol{\Sigma}_t := \mathrm{diag}(1/(N(\mathbf{s}_t + \tilde{\delta})))$ using the following update (element-wise):

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

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

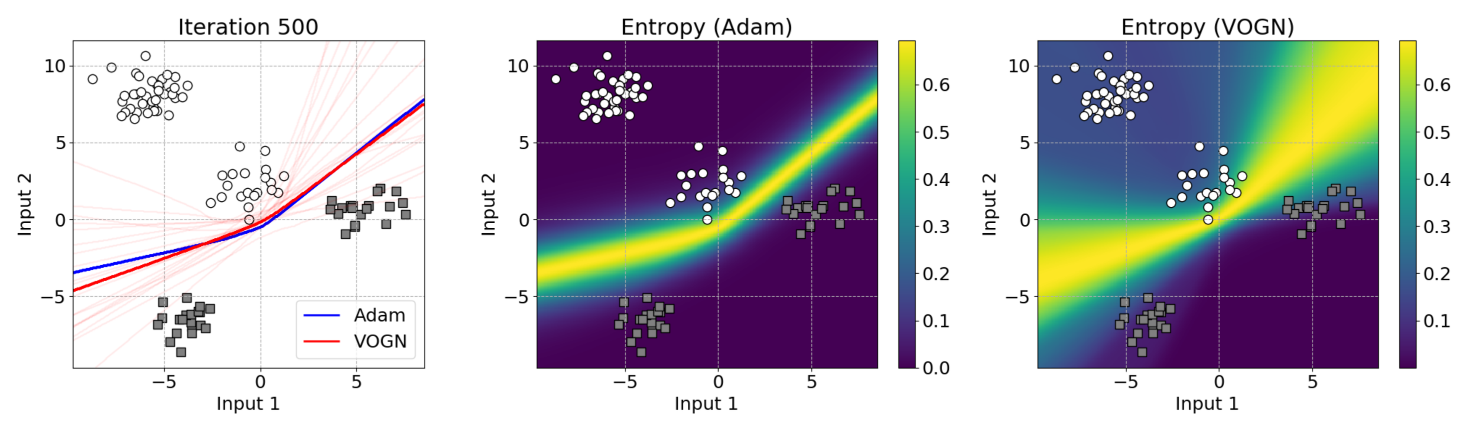


Figure: Decision boundary and entropy plots on 2D-binary classification by MLPs trained with Adam and VOGN. VOGN optimises the posterior distribution of each weight (i.e., mean and variance of the Gaussian). A model with the mean weights draws the red boundary, and models with the MC samples from the posterior distribution draw light red boundaries. VOGN converges to a similar solution as Adam while keeping uncertainty in its predictions.

NGVI + Deep Learning Techniques

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

- Batch normalization
- Data augmentation
- Momentum and initialisation
- Learning rate scheduling

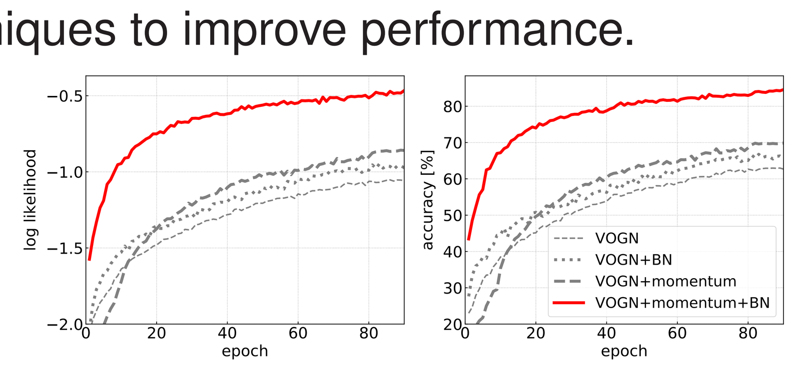


Figure: Effect of momentum and batch normalisation. (ResNet-18 on CIFAR-10)

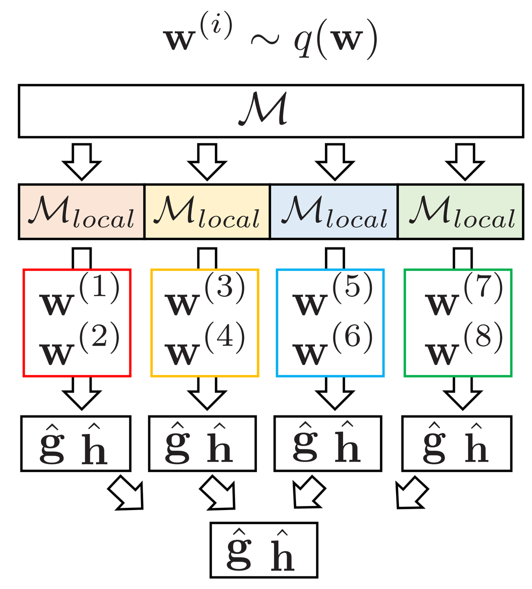
NGVI + Distributed Deep Learning

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.
- **MC sample parallelism**: different GPUs use different MC samples from the posterior for stabilising the training.

Algorithm 1 Distributed VOGN

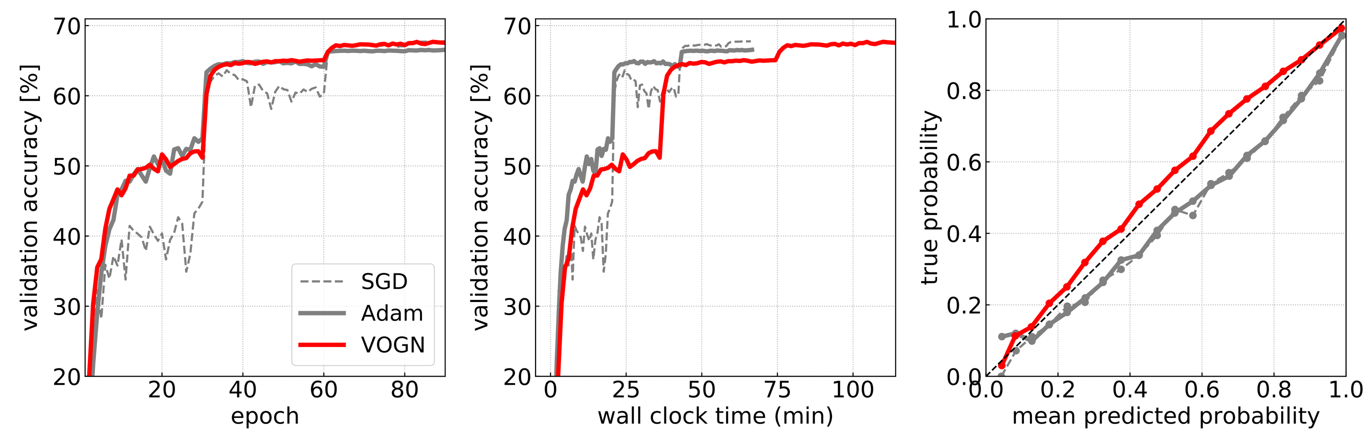
Initialise $\boldsymbol{\mu}_0$, $\mathbf{s}_0$, $\mathbf{m}_0$.
$N \leftarrow \rho N$, $\tilde{\delta} \leftarrow \tau\delta/N$.
repeat
  Sample a minibatch $\mathcal{M}$ of size M.
  Split $\mathcal{M}$ into each GPU (local minibatch $\mathcal{M}_{local}$).
  for each GPU in parallel **do**
    for $k = 1, 2, \ldots, K$ **do**
      Sample $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
      $\mathbf{w}^{(k)} \leftarrow \boldsymbol{\mu} + \epsilon\boldsymbol{\sigma}$ with $\boldsymbol{\sigma} \leftarrow (1/(N(\mathbf{s} + \tilde{\delta} + \gamma)))^{1/2}$.
      Compute $\mathbf{g}_i^{(k)} \leftarrow \nabla_w \ell(\mathbf{y}_i, \mathbf{f}_{w^{(k)}}(\mathbf{x}_i)), \forall i \in \mathcal{M}_{local}$
      $\hat{\mathbf{g}}_k \leftarrow \frac{1}{M}\sum_{i \in \mathcal{M}_{local}} \mathbf{g}_i^{(k)}$.
      $\hat{\mathbf{h}}_k \leftarrow \frac{1}{M}\sum_{i \in \mathcal{M}_{local}} (\mathbf{g}_i^{(k)})^2$.
    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.
  $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (\hat{\mathbf{g}} + \tilde{\delta}\boldsymbol{\mu})$.
  $\mathbf{s} \leftarrow (1 - \tau\beta_2)\mathbf{s} + \beta_2 \hat{\mathbf{h}}$.
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha\mathbf{m}/(\mathbf{s} + \tilde{\delta} + \gamma)$.
until stopping criterion is met =0



Learning rate	α
Momentum rate	β_1
Exp. moving average rate	β_2
Prior precision	δ
External damping factor	γ
Tempering parameter	τ
# MC samples for training	K
Data augmentation factor	ρ

Large-Scale VI on ImageNet classification

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



Performance comparisons on different dataset/architecture

Dataset/ Architecture	Optimiser	Train/Validation Accuracy (%)	Validation NLL	Epochs	Time/ epoch (s)	ECE	AUROC
CIFAR-10/ LeNet-5 (no DA)	Adam	71.98 / **67.67**	**0.937**	210	6.96	**0.021**	0.794
	BBB	66.84 / 64.61	1.018	800	11.43†	0.045	0.784
	MC-dropout	68.41 / **67.65**	0.99	210	6.95	0.087	**0.797**
	VOGN	70.79 / **67.32**	**0.938**	210	18.33	0.046	**0.8**
CIFAR-10/ AlexNet (no DA)	Adam	100.0 / 67.94	2.83	161	3.12	0.262	0.793
	MC-dropout	97.56 / **72.20**	1.077	160	3.25	0.140	**0.818**
	VOGN	79.07 / 69.03	**0.93**	160	9.98	**0.024**	0.796
CIFAR-10/ AlexNet	Adam	97.92 / 73.59	1.480	161	3.08	0.262	0.793
	MC-dropout	80.65 / **77.04**	**0.667**	160	3.20	0.114	0.828
	VOGN	81.15 / 75.48	0.703	160	10.02	**0.016**	**0.832**
CIFAR-10/ ResNet-18	Adam	97.74 / **86.00**	0.55	160	11.97	0.082	**0.877**
	MC-dropout	88.23 / 82.85	0.51	161	12.51	0.166	0.768
	VOGN	91.62 / 84.27	**0.477**	161	53.14	**0.040**	**0.876**
ImageNet/ ResNet-18	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
	MC-dropout	72.96 / 65.64	1.43	90	45.86	**0.012**	0.856
	OGN	85.33 / 65.76	1.60	90	63.13	0.128	0.854
	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.

PyTorch Implementation

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

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.