[Paper review 38]

Practical Deep Learning with Bayesian Principles

(Osawa, et al., 2019)

[Contents]

1. Abstract
2. Introduction
3. Deep Learning with Bayesian Principles & its Challenges
4. Practical Deep Learning with Natural-Gradient Variational Inference
5. Algorithm

1. Abstract

Bayesian

- pros) can fix many shortcomings of DL
- cons) impractical

This paper shows "practical training of DNN with **natural gradient variational inference**"

- use batch norm / data augmentation / distributed training / ....
- similar performance as Adam (even on large datasets)

Keeps benefits of Bayesian view

- 1) predictive probabilities are well calibrated
- 2) uncertainties of OOD is improved
- 3) continual-learning performance is boosted

2. Introduction

problem of DL

- 1) need large dataset (o.w, overfit!)
- 2) sequential learning can cause forgetting of past knowledge
- 3) lack of uncertainty estimation

Bayesian can solve those 3 problems!

- 1) using Bayesian model averaging
- 2) sequential learning with Bayes' rule
- 3) uncertainty estimation with posterior distribution

(previous) Bayesian inference on NN

- MCMC
- Laplace's method
- VI

$\rightarrow$ rarely used, due to **impracticality** (because of **computation of posterior**)

New approaches ex)

- MC-dropout (***Dropout as a Bayesian approximation***) : less-principled, but unsuitable for continual learning

Goal of this paper : **"make VI practical"**

"By using recently-proposed **natural gradient VI** method"

3. Deep Learning with Bayesian Principles & its Challenges

form of loss function : $\bar{\ell}(\mathbf{w})+\delta \mathbf{w}^{\top} \mathbf{w}$

- where $\bar{\ell}(\mathbf{w}):=\frac{1}{N} \sum_{i} \ell\left(\mathbf{y}_{i}, \mathbf{f}_{w}\left(\mathbf{x}_{i}\right)\right), \mathbf{f}_{w}(\mathbf{x}) \in \mathbb{R}^{K}$

widely used optimizers

- ex) SGD,RMSprop, Adam...
- form : $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_{t}-\alpha_{t} \frac{\hat{\mathrm{g}}\left(\mathbf{w}_{t}\right)+\delta \mathbf{w}_{t}}{\sqrt{\mathbf{s}_{t+1}}+\epsilon}, \quad \mathbf{s}_{t+1} \leftarrow\left(1-\beta_{t}\right) \mathbf{s}_{t}+\beta_{t}\left(\hat{\mathbf{g}}\left(\mathbf{w}_{t}\right)+\delta \mathbf{w}_{t}\right)^{2}$
  - t : iteration
  - $\alpha_{t}>0$ and $0<\beta_{t}<1$: learning rates
  - $\hat{\mathrm{g}}(\mathrm{w})$: stochastic gradients at $\mathbf{w}$

    ($\hat{\mathrm{g}}(\mathbf{w}):=\frac{1}{M} \sum_{i \in \mathcal{M}_{t}} \nabla_{w} \ell\left(\mathbf{y}_{i}, \mathbf{f}_{w}\left(\mathbf{x}_{i}\right)\right)$ ... using minibatch)

  $\rightarrow$ scales extremely well!

full Bayesian approach

- computationally very expensive in DNN
- use Bayes rule to get posterior

  ($p(\mathbf{w} \mid \mathcal{D})=\exp (-N \bar{\ell}(\mathbf{w}) / \tau) p(\mathbf{w}) / p(\mathcal{D})$)

VI

- principled approach, to more scalably estimate an approximation to posterior $p(\mathbf{w} \mid \mathcal{D})$
- $q(\mathbf{w}) := \mathcal{N}(\mathbf{w} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$
- optimization by maximizing ELBO
  ($\text{ELBO:} \quad \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) := -N\mathbb{E}_q[\bar{\ell}(\mathbf{w})] - \tau\mathbb{D}_{KL}[q(\mathbf{w}) \| p(\mathbf{w})]$)
- the more complex, the more computational cost

$\rightarrow$ have still remained impractical....

4. Practical Deep Learning with Natual-Gradient Variational Inference

propose natural-gradient VI methods

- simple, when estimating **exponential-family approximation**
- ex) when $p(\mathbf{w}) := \mathcal{N}(\mathbf{w} \mid 0, \mathbf{I}/\delta)$,.
  update of natural-parameter λ : $\boldsymbol{\lambda}_{t+1} = (1 - \tau\rho)\boldsymbol{\lambda}_t - \rho\nabla_\mu\mathbb{E}_q\left[\bar{\ell}(\mathbf{w}) + \frac{1}{2}\tau\delta\mathbf{w}^\top\mathbf{w}\right].$
  - ρ : learning rate
  - use **moving average**
    (if $\tau = 0$, update = minimize the regularized loss)

VOGN (Variational Online Gauss-Newton) method

- if q : Gaussian..
  - $\boldsymbol{\lambda}_{t+1} = (1 - \tau\rho)\boldsymbol{\lambda}_t - \rho\nabla_\mu\mathbb{E}_q\left[\bar{\ell}(\mathbf{w}) + \frac{1}{2}\tau\delta\mathbf{w}^\top\mathbf{w}\right]$ is becomes similar to
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha_t\frac{\hat{\mathrm{g}}(\mathbf{w}_t) + \delta\mathbf{w}_t}{\sqrt{\mathbf{s}_{t+1}} + \epsilon}, \quad \mathbf{s}_{t+1} \leftarrow (1 - \beta_t)\,\mathbf{s}_t + \beta_t(\hat{\mathbf{g}}(\mathbf{w}_t) + \delta\mathbf{w}_t)^2$
- $\mu_{t+1} \leftarrow \mu_t - \alpha_t\frac{\hat{\mathrm{g}}(\mathrm{w}_t) + \tilde{\delta}\mu_t}{\mathrm{s}_{t+1} + \tilde{\delta}}, \quad \mathrm{s}_{t+1} \leftarrow (1 - \tau\beta_t)\,\mathrm{s}_t + \beta_t\frac{1}{M}\sum_{i \in \mathcal{M}_t}(\,\mathrm{g}_i\,(\mathrm{w}_t))^2.$
  - $\mathbf{g}_i\,(\mathbf{w}_t) := \nabla_w\ell\,(y_i, f_{w_t}\,(\mathbf{x}_i)).$
    $\mathbf{w}_t \sim \mathcal{N}\,(\mathbf{w} \mid \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ where $\boldsymbol{\Sigma}_t := \operatorname{diag}\big(1/\left(N\left(\mathbf{s}_t + \tilde{\delta}\right)\right)\big), \tilde{\delta} := \tau\delta/N$
  - s_t : adapts the learning rate (updated using moving average)
- major difference:
  - update of s_t is based on Gauss-Newton approximation
    ($\frac{1}{M}\sum_{i \in \mathcal{M}_t}(\mathbf{g}_i\,(\mathbf{w}_t))^2$)

This paper :

- show practical training of DNN using **VOGN**

Techniques

- (1) Batch Norm
  - speed up & stabilize training

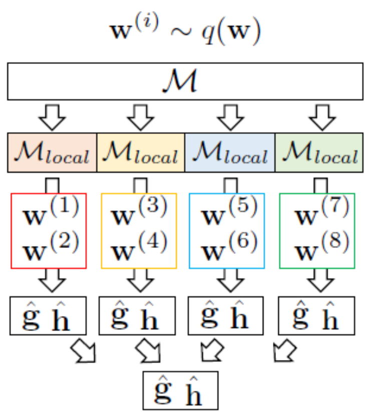
    - do not use L2 reg/ weight decay
- (2) Data Augmentation
    - improve performance drastically
- (3) Momentum & initialization
    - to improve speed of convergence
    - β_1 : momentum rate
    - $\mathbf{m}$: momentum term
    - Xavier initialization
- (4) Learning rate scheduling
    - regularly decayed
- (5) Distributed training
    - to perform large experiments quickly
    - parallelize (over data & MC samples)
    - reduce variance
- (6) Implementation of Gauss-Newton update in VOGN
    - use Gauss-Newton approximation

      (different from Adam!)

5. Algorithm

Algorithm 1: Variational Online Gauss Newton (VOGN)

```
1:  Initialise $\mu_0, \mathbf{s}_0, \mathbf{m}_0$.
2:  $N \leftarrow \rho N, \tilde{\delta} \leftarrow \tau\delta/N$.
3:  repeat
4:      Sample a minibatch $\mathcal{M}$ of size $M$.
5:      Split $\mathcal{M}$ into each GPU (local minibatch $\mathcal{M}_{local}$).
6:      for each GPU in parallel do
7:          for $k = 1, 2, \ldots, K$ do
8:              Sample $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
9:              $\mathbf{w}^{(k)} \leftarrow \mu + \epsilon\sigma$ with $\sigma \leftarrow (1/(N(\mathbf{s} + \tilde{\delta} + \gamma)))^{1/2}$.
10:             Compute $\mathbf{g}_i^{(k)} \leftarrow \nabla_w \ell(\mathbf{y}_i, \mathbf{f}_{w^{(k)}}(\mathbf{x}_i)), \forall i \in \mathcal{M}_{local}$
                using the method described in Appendix B.
11:             $\hat{\mathbf{g}}_k \leftarrow \frac{1}{M} \sum_{i \in \mathcal{M}_{local}} \mathbf{g}_i^{(k)}$.
12:             $\hat{\mathbf{h}}_k \leftarrow \frac{1}{M} \sum_{i \in \mathcal{M}_{local}} (\mathbf{g}_i^{(k)})^2$ .
13:         end for
14:         $\hat{\mathbf{g}} \leftarrow \frac{1}{K} \sum_{k=1}^{K} \hat{\mathbf{g}}_k$ and $\hat{\mathbf{h}} \leftarrow \frac{1}{K} \sum_{k=1}^{K} \hat{\mathbf{h}}_k$.
15:     end for
16:     AllReduce $\hat{\mathbf{g}}, \hat{\mathbf{h}}$.
17:     $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (\hat{\mathbf{g}} + \tilde{\delta}\mu)$.
18:     $\mathbf{s} \leftarrow (1 - \tau\beta_2)\mathbf{s} + \beta_2 \hat{\mathbf{h}}$.
19:     $\mu \leftarrow \mu - \alpha\mathbf{m}/(\mathbf{s} + \tilde{\delta} + \gamma)$.
20: until stopping criterion is met
```


$\mathbf{w}^{(i)} \sim q(\mathbf{w})$
$\mathcal{M}$
$\mathcal{M}_{local}$
$\mathcal{M}_{local}$
$\mathcal{M}_{local}$
$\mathcal{M}_{local}$
$\mathbf{w}^{(1)}$ $\mathbf{w}^{(2)}$
$\mathbf{w}^{(3)}$ $\mathbf{w}^{(4)}$
$\mathbf{w}^{(5)}$ $\mathbf{w}^{(6)}$
$\mathbf{w}^{(7)}$ $\mathbf{w}^{(8)}$
$\hat{\mathbf{g}}$ $\hat{\mathbf{h}}$

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

Figure 2: A pseudo-code for our distributed VOGN algorithm is shown in Algorithm 1, and the distributed scheme is shown in the right figure. The computation in line 10 requires an extra calculation (see Appendix B), making VOGN slower than Adam. The bottom table gives a list of algorithmic hyperparameters needed for VOGN.