[Paper review 14]

Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

(B. Lakshminarayanan et al., 2017)

[Contents]

0. Abstract

Bayesian NN : SOTA for estimating predictive uncertainty

Propose an alternative to BNN!

- simple to implement
- parallelizable
- requires very little hyperparamter tuning
- yields high quality predictive uncertainty estimates

Better than approximate BNNs!

1. Introduction

focus on 2 evaluation measures

- 1) calibration
- 2) generalization to unknown class

Calibration

- discrepancy between subjective forecasts & (empirical) long run frequencies
- can be measured by "proper scoring rules"

Generalization to unknown class

- generalization of the predictive uncertainty to domain shift (= out-of-domain examples)
- "measuring if the network KNOWS what it KNOWS"

  ex) if a network (trained on one dataset) is evaluated on completely different dataset, should output high predictive uncertainty!

Summary of contributions

- 1) describe "simple & scalable method for estimating predictive uncertainty estimates from NNs"

  (using proper scoring rule)

  (+ two modifications : (1) ensembles & (2) adversarial training)
- 2) evaluating the quality of thee predictive uncertainty

  (in terms of (1) calibration & (2) generalization to unknown classes)

Out performs MCDO (Monte Carlo Drop Out) !!

Novelty and Significance

- (1) Ensembles of NN (=deep ensembles) : boost performance
- (2) Adversarial training : improve robustness
- first work to investigate that (1) & (2) can be useful for predictive uncertainty estimation!

2. Deep Ensembles :

A simple Recipe for Predictive Uncertainty Estimation

2.1 Problem setup & High-level summary

(Very Simple!)

(step 1) use a proper scoring rule as a training criterion

(step 2) use adversarial training to smooth the predictive distributions

(step 3) train an ensemble

2.2 Proper Scoring Rules

scoring rule

- function $S\left(p_\theta,(y, \mathbf{x})\right)$
- evaluates the quality of the predictive distribution $p_\theta(y \mid \mathbf{x})$,
  relative to an event $y \mid \mathrm{x} \sim q(y \mid \mathrm{x})$ (where $q(y, \mathrm{x})$ is a true distribution)
- the higher , the better

proper scoring rules

- one where $S\left(p_\theta, q\right) \leq S(q, q)$

  with equality if and only if $p_\theta(y \mid \mathrm{x})=q(y \mid \mathrm{x})$, for all p_θ and q
- then, NNs are trained by minimizing the $\operatorname{loss} \mathcal{L}(\theta)=-S\left(p_\theta, q\right)$

  (encourages calibration of predictive uncertainty)

Examples

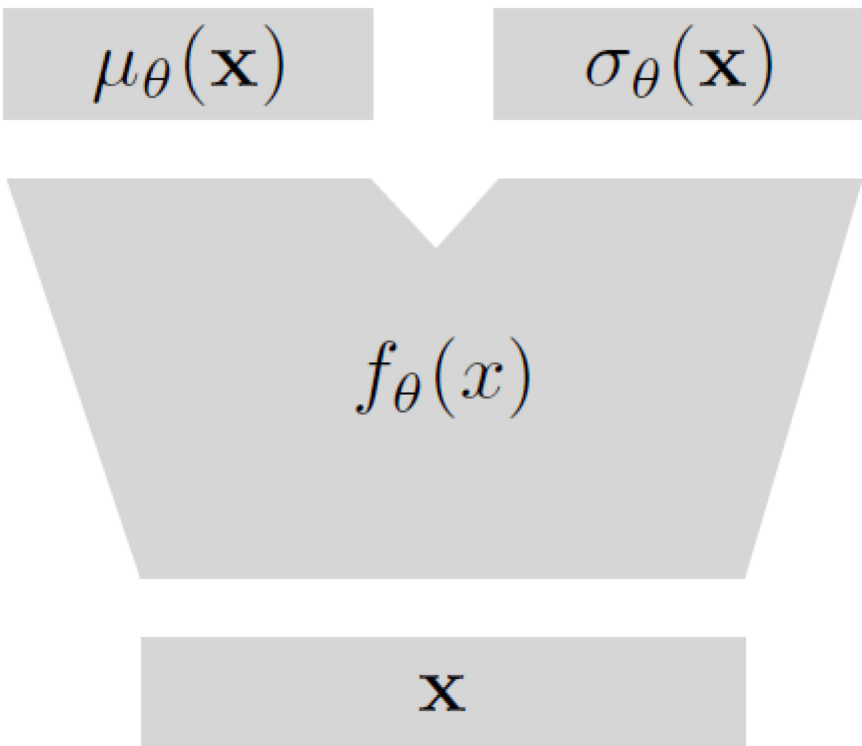
- maximizing MLE : $S\left(p_\theta,(y, \mathbf{x})\right)=\log p_\theta(y \mid \mathbf{x})$,
- softmax (cross entropy) loss : log likelihood
- minimizing the squared error : $\mathcal{L}(\theta)=-S\left(p_\theta,(y, \mathbf{x})\right)=K^{-1} \sum_{k=1}^K\left(\delta_{k=y}-p_\theta(y=k \mid \mathrm{x})\right)^2$

2.2.1 Training criterion for regression

MSE : does not capture predictive uncertainty

Use network with 2 output values (in final layer)

- predicted mean $\mu(x)$
- predicted variance $\sigma^2(x)$



Treat observed samples from Gaussian (with predicted mean & variance)

That is, we minimize NLL criterion.

$\mathcal{L}=-\frac{1}{N} \sum_{i=1}^{N} \log \mathcal{N}\left(y_{i} ; \mu_{\theta}\left(x_{i}\right), \sigma_{\theta}^{2}\left(x_{i}\right)\right)$

$-\log p_{\theta}\left(y_{n} \mid \mathbf{x}_{n}\right)=\frac{\log \sigma_{\theta}^{2}(\mathbf{x})}{2}+\frac{\left(y-\mu_{\theta}(\mathbf{x})\right)^{2}}{2 \sigma_{\theta}^{2}(\mathbf{x})}+C$

2.3 Adversarial training to smooth predictive distributions

Adversarial examples : 'close' to the original training examples, but are misclassified by NN

Fast gradient sign method (Goodfellow et al.)

- way to generate adversarial example
- $\mathbf{x}^{\prime}=\mathbf{x}+\epsilon \operatorname{sign}\left(\nabla_{\mathbf{x}} \ell(\theta, \mathbf{x}, y)\right)$

Adversarial perturbation "creates a new training example" by adding a perturbation along a direction "which the network is likely to increase loss"

if ϵ is small enough

- can be used to augment the original training set!

  (by treating $(x^{\prime}, y)$ as additional samples)
- improve classifier's robustness!

Interestingly, adversarial training can be interpreted as a computationally efficient solution to smooth the predictive distributions by increasing the likelihood of the target around an ϵ-neighborhood of the observed training examples.

2.4 Ensembles

Bagging & Boosting

Bagging

- with complex model
- reduce variance

Boosting

- with simple model
- reduce bias

3. Algorithm

Algorithm 1 Pseudocode of the training procedure for our method

$\triangleright$ *Let each neural network parametrize a distribution over the outputs, i.e.* $p_\theta(y|\mathbf{x})$. *Use a proper scoring rule as the training criterion* $\ell(\theta, \mathbf{x}, y)$. *Recommended default values are* $M = 5$ *and* $\epsilon = 1\%$ *of the input range of the corresponding dimension (e.g 2.55 if input range is [0,255]).*
Initialize $\theta_1, \theta_2, \ldots, \theta_M$ randomly
for $m = 1 : M$ **do** $\triangleright$ *train networks independently in parallel*
  Sample data point n_m randomly for each net $\triangleright$ *single* n_m *for clarity, minibatch in practice*
  Generate adversarial example using $\mathbf{x}'_{n_m} = \mathbf{x}_{n_m} + \epsilon \operatorname{sign}\left(\nabla_{\mathbf{x}_{n_m}} \ell(\theta_m, \mathbf{x}_{n_m}, y_{n_m})\right)$
  Minimize $\ell(\theta_m, \mathbf{x}_{n_m}, y_{n_m}) + \ell(\theta_m, \mathbf{x}'_{n_m}, y_{n_m})$ w.r.t. θ_m $\triangleright$ *adversarial training (optional)*

combine predictions!

$p(y \mid \mathrm{x}) = M^{-1} \sum_{m=1}^{M} p_{\theta_m}(y \mid \mathrm{x}, \theta_m)$

- classification) averaging the predicted probabilities.
- regression) mixture of Gaussian distributions

Approximate the ensemble prediction as a Gaussian

$p(y \mid \mathrm{x}) = M^{-1} \sum_{m=1}^{M} p_{\theta_m}(y \mid \mathrm{x}, \theta_m) \approx M^{-1} \sum \mathcal{N}\left(\mu_{\theta_m}(\mathrm{x}), \sigma^2_{\theta_m}(\mathrm{x})\right)$

- mean : $\mu_*(\mathrm{x}) = M^{-1} \sum_m \mu_{\theta_m}(\mathrm{x})$
- variance : $\sigma^2_*(\mathrm{x}) = M^{-1} \sum_m \left(\sigma^2_{\theta_m}(\mathrm{x}) + \mu^2_{\theta_m}(\mathrm{x})\right) - \mu^2_*(\mathrm{x})$