Summary

He Li

2017/8/12

I. Paper Review

1. Mean and Variance Method

- Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2016). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. arXiv preprint arXiv:1612.01474

Two measures of the quality of predictive uncertainty

1. Calibration: the discrepancy between subjective forecasts and empirical long-run frequencies, the calibration should be measured by *proper scoring rules* e.g. log likelihood and Brier score.
2. Generalization of the predictive uncertainty: when shifting to a different dataset, the network should output high predictive uncertainty.

Main idea for uncertainty estimation

1. use a proper scoring rule as the training criterion
2. use adversarial training to smooth the predictive distributions
3. train an ensemble

proper scoring rule

Using the MSE as cost function does not capture predictive uncertainty. Instead, we minimise the negative log-likelihood criterion:

$$-log p_\theta(y_n|x_n) = \frac{log \sigma_\theta^2(x)}{2} + \frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)} + C$$

Here, in cases where the Gaussian is too-restrictive, one could use a complex distribution e.g. mixture density network.

adversarial training

Using the fast gradient sign method, We can generate an adversarial example as:

$$x' = x + \epsilon \text{sign}\left(\bigtriangledown_\theta \mathcal{L}(\theta, x, y)\right)$$

where ϵ is a small value such that the max-norm of the perturbation is bounded. Adversarial training can be interpreted as a computationally efficient solution to smooth the predictive dis tributions by increasing the likelihood of the target around an ϵ-neighborhood of the observed training examples.

training

The overall training procedure is as follows:

1. Similar to the mean and variance methods, we train M network to estimate mean and variance together, each network is initialized randomly using the entire training dataset.
2. During the backpropogation process, for each mini-batch example, we generate its adversarial example and calculate gradient using both.
3. Estimate on the test set, for each example x, we get $\mu_1(x), ..., \mu_M(x)$ and $\sigma_1^2(x), ..., \sigma_M^2(x)$. Then,

$$
\begin{aligned}
\hat{\mu}(x) &= \frac{1}{M}\sum_i \mu_i(x) \\
\hat{\sigma}^2(x) &= \frac{1}{M}\sum_i \left(\sigma_i^2(x) + \mu_i^2(x)\right) - \hat{\mu}^2(x)
\end{aligned}
$$

2. Bayesian Methods

- Gal, Y., & Ghahramani, Z. (2016, June). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059)

Main Idea

Deep neural network with arbitrary depth and non-linearities, with dropout applied before every weight layer, is mathematically equivalent to an approximation to the probabilistic deep Gaussian process. The proof is conducted without any simplification. Furthermore, the results carry to other variants of dropout like drop-connect, multiplicative Gaussian noise.

- Li, Y., & Gal, Y. (2017). Dropout Inference in Bayesian Neural Networks with Alpha-divergences. arXiv preprint arXiv:1703.02914

3. Calibration

- Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks. arXiv preprint arXiv:1706.04599

Confidence calibration is the problem of predicting probability estimates representative of the true correctness likelihood. Traditional neural networks typically produce well-calibrated probabilities on binary classification tasks. While neural networks today no longer well-calibrated. Here perfect calibration is defined as:

$$P\left(\hat{Y} = Y | \hat{P} = p\right) = p$$

Measure of confidence calibration

Reliability Diagrams: visual representation of model calibration, plot expected sample accuracy as a function of confidence. Group predictions into M interval bins with size $\frac{1}{M}$, and calculate the accuracy of each bin. Let B_m be the set of indices of samples whose prediction confidence falls into the interval $I_m = (\frac{m-1}{M}, \frac{m}{M}]$. The accuracy of B_m is:

$$\mathrm{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbf{1}(\hat{y}_i = y_i)$$

here $acc(B_m)$ is unbiased and consistent estimator of $P\left(\hat{Y} = Y | \hat{P} \in I_m\right)$. We define the average confidence within bin B_m as:

$$\mathrm{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i$$

therefore, a perfectly calibrated model will have $acc(B_m) = conf(B_m)$.

Expected Calibration Error (ECE): One notion of miscalibration is the difference in expectation between confidence and accuracy

$$\mathbb{E}_{\hat{P}}\left[\left|P\left(\hat{Y} = Y | \hat{P} = p\right) - p\right|\right]$$

estimate the above function using ECE:

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left|\mathrm{acc}(B_m) - \mathrm{conf}(B_m)\right|$$

Maximum Calibration Error (MCE)

In high-risk applications, wish to minimize the worst-case deviation between confidence and accuracy:

$$\max_{p \in [0,1]} \left|P\left(\hat{Y} = Y | \hat{P} = p\right) - p\right|$$

estimate the above function using MCE:

$$MCE = \max_{m \in \{1,\ldots,M\}} \left|acc(B_m) - conf(B_m)\right|$$

Observing Miscalibration

1. **Model capacity**: increasing depth and width may increases negatively affect model calibration.
2. **Batch Normalization**: models trained with Batch Normalization tend to be more miscalibrated.
3. **Weight decay**: training with less weight decay has a negative impact on calibration.
4. **NLL**: observe a disconnect between NLL and accuracy, which may explain the miscalibration above. This disconnect occurs because neural networks can overfit to NLL without overfitting to the 0/1 loss.

Calibrating Models

Histogram binning: define bin boundaries $0 = a_1 \leq a_2 \leq \ldots \leq a_{M+1} = 1$, then predictions θ_i are chosen to minimize the bin-wise squared loss:

$$\min_{\theta_1,\ldots,\theta_M} \sum_{m=1}^{M} \sum_{i=1}^{n} \mathbf{1}(a_m \leq \hat{p}_i \leq a_{m+1})(\theta_m - y_i)^2$$

where $\mathbf{1}$ is the indicator function and θ_m some calibrated score assigned to each bin.

Isotonic regression: the most common non-parametric calibration method, which learns a piecewise constant function f to transform uncalibrated outputs

$$\min_{\substack{M \\ \theta_1,\ldots,\theta_M \\ a_1,\ldots,a_{M+1}}} \sum_{m=1}^{M} \sum_{i=1}^{n} \mathbf{1}(a_m \leq \hat{p}_i \leq a_{m+1})(\theta_m - y_i)^2$$

here θ_m denotes the piecewise function values. Under this parameterization, isotonic regression is a strict generalization of histogram binning in which the bin boundaries and bin predictions are jointly optimized.

Platt scaling: parametric approach parametric approach, learns scalar parameters $a, b \in \mathbb{R}$ and outputs $\hat{q}_i = \sigma(az_i + b)$ as the calibrated probability. Parameters a and b can be optimized using the NLL loss over the validation set. We can extend the binary problem into multi-class one by change a, b to W, b, a.k.a. matrix scaling.

Temperature scaling: the simplest extension of Platt scaling, uses a single scalar parameter $T > 0$ for all classes. Given the logit vector z_i, the new confidence prediction is:

$$\hat{q}_i = \max_k \sigma(\frac{z_i}{T})^{(k)}$$

T is called the temperature, and it softens the softmax (i.e. raises the output entropy) with $T > 1$. T is optimized with respect to NLL on the validation set.

- Niculescu-Mizil, A., & Caruana, R. (2005, August). Predicting good probabilities with supervised learning. In Proceedings of the 22nd international conference on Machine learning (pp. 625-632). ACM

4. Estimation for Logistic-Normal Integrals

- Crouch, E. A., & Spiegelman, D. (1990). The Evaluation of Integrals of the form +∞−∞ f (t) exp (− t 2) dt: Application to Logistic-Normal Models. Journal of the American Statistical Association, 85(410), 464-469

We can extimate the logistic-normal integrals with the general form $\int_{-\infty}^{\infty} f(t)\ exp\{-t^2\}dt$ by:

$$\begin{aligned}\int_{-\infty}^{\infty} f(t)\ exp\{-t^2\}dt = \varepsilon(h)\\ + h\sum_{n=-\infty}^{\infty} f(t_0+nh)\ exp\left[-(t_0+nh)^2\right]\\ + 2\pi i \sum_{below} res\left\{\frac{f(t_j)exp(-t_j^2)exp[-2\pi i(t-t_0)/h]}{1-exp[-2\pi i(t_j-t_0)/h]}\right\}\\ + 2\pi i \sum_{above} res\left\{\frac{f(t_j)exp(-t_j^2)}{1-exp[-2\pi i(t_j-t_0)/h]}\right\}\end{aligned}$$

where the summations over the residues of the poles t_j of $f(t)$ are over those below and above the real axis, respectively, in the strip $-k < \mathcal{F}(t) < k$ and

$$|\varepsilon(h)| \leq exp(k^2 - 2\frac{\pi k}{h})\int_{-\infty}^{\infty}\{|f(t+ik)| + |f(t-ik)|\}\ exp(-t^2)\ dt$$

here we can choose k, h such that the overall error term $\varepsilon(h)$ is as small as desired while h is as large as posiible to truncate the infinite sum term after few terms.

5. Other Methods

- Koh, P. W., & Liang, P. (2017). Understanding black-box predictions via influence functions. arXiv preprint arXiv:1703.04730

Main Idea

1. Measure the impact of a training point on the prediction: influence function
2. estimate influence function: second-order optimization techniques

Approach

Upweighting a training point

How to measure model's predictions change without one training point? Foramlly, $\hat{\theta_{-z}} - \hat{\theta}$ where

$$\hat{\theta_{-z}} \triangleq \underset{\theta\in\Theta}{argmax} \sum_{z_i\neq z} L(z_i, \theta)$$

here we can use influence function to approximation. The idea is to compute the parameter change if z were upweighted by some small ϵ. The $\hat{\theta}_{\epsilon,z}$ is given by:

$$\hat{\theta}_{\epsilon,z} \triangleq \underset{\theta\in\Theta}{argmax}\, \frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta)\ + \epsilon L(z,\theta)$$

then the influence of upweighting z on the parameters $\hat{\theta}$ is given by:

$$\mathscr{L}_{up,param}(z) \triangleq \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1} \bigtriangledown_{\theta} L(z,\hat{\theta})$$

where H is the Hessian, assumed PD and can be estimated by $H_{\hat{\theta}} = \frac{1}{n}\sum_{i=1}^{n} \bigtriangledown_{\theta}^{2} L(z_i,\theta)$. Therefore, we can linearly approximate the parameter change due to removing z by:

$$\hat{\theta}_{-z} - \hat{\theta} \approx -\frac{1}{n}\mathscr{L}_{up,param}(z)$$

the influence of upweighting z on the loss at a test point z_{test}:

$$\mathscr{L}_{up,loss}(z,z_{test}) \triangleq \left.\frac{dL\left(z_{test},\hat{\theta}_{\epsilon,z}\right)}{d\epsilon}\right|_{\epsilon=0} = -\bigtriangledown_{\theta} L(z_{test},\hat{\theta})^{\top} H_{\hat{\theta}}^{-1} \bigtriangledown_{\theta} L(z,\hat{\theta})$$

Perturbing a training input

How to measure model's predictions change with one training point perturbed? Here, given $z = (x,y)$, the pertubation means $z_{\delta} = (x+\delta,y)$. Then define the new parameters is:

$$\hat{\theta}_{\epsilon,z_{\delta}} \triangleq \underset{\theta\in\Theta}{argmax}\, \frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta)\ + \epsilon L(z_{\theta},\theta) - \epsilon L(z,\theta)$$

Similarly,

$$\begin{aligned}\mathscr{L}_{pert,param}(z) \triangleq \left.\frac{d\hat{\theta_{\epsilon,z_{\delta}}}}{d\epsilon}\right|_{\epsilon=0} &= \mathscr{L}_{up,param}(z_{\delta}) - \mathscr{L}_{up,param}(z) \\ &= -H_{\hat{\theta}}^{-1}\left(\bigtriangledown_{\theta} L(z_{\delta},\hat{\theta}) - \bigtriangledown_{\theta} L(z,\hat{\theta})\right)\end{aligned}$$

Then the linear approximation is:

$$\hat{\theta}_{z_{\delta}} - \hat{\theta} \approx -\frac{1}{n}\left(\mathscr{L}_{up,param}(z_{\delta}) - \mathscr{L}_{up,param}(z)\right)$$

If here x is continuous, as is the case in most situations, we can make furthur approximation:

$$\left.\frac{d\hat{\theta}_{\epsilon,z_{\delta}}}{d\epsilon}\right|_{\epsilon=0} \approx -H_{\hat{\theta}}^{-1}\left[\bigtriangledown_{x} \bigtriangledown_{\theta} L(z,\hat{\theta})\right]\delta$$

and

$$\hat{\theta}_{z_{\delta}} - \hat{\theta} \approx -\frac{1}{n} H_{\hat{\theta}}^{-1}\left[\bigtriangledown_{x} \bigtriangledown_{\theta} L(z,\hat{\theta})\right]\delta$$

the influence of pertubing z on the loss at a test point z_{test}:

$$\mathscr{L}_{pert,loss}(z,z_{test})^{\top} \triangleq \left.\frac{\partial L\left(z_{test},\hat{\theta}_{z_{\delta}}\right)}{\partial\delta}^{\top}\right|_{\delta=0} = -\bigtriangledown_{\theta} L(z_{test},\hat{\theta})^{\top} H_{\hat{\theta}}^{-1}\left(\bigtriangledown_{x} \bigtriangledown_{\theta} L(z,\hat{\theta})\right)$$

Efficiency Estimation

- Computation on Hessian: use stochastic estimation, by only sampling a single point per iteration
- Non-convexity and non-convergence: form a convex quadratic approximation of the loss around $\tilde{\theta}$:

$$\tilde{L}(z,\theta) = L(z,\tilde{\theta}) + \bigtriangledown L(z,\tilde{\theta})^\top(\theta-\tilde{\theta}) + \frac{1}{2}(\theta-\tilde{\theta})^\top(H_{\tilde{\theta}}+\lambda I)(\theta-\tilde{\theta})$$

- Non-differentiable losses: smoothen the non-linear functions

Application

- Understanding model behavior
- Adversarial training: influence functions can be used to craft adversarial training images that are similarly visually-indistinguishable and can flip a model's prediction on a separate test image
- Fixing mislabeled: flag the training points that exert the most influence on the model

II. Ideas

1. Mean & Variance Estimation

Model

Write our likelihood function in the following form:

$$p(y|\mu;\sigma^2) = \int_{-\infty}^{\infty} p(y|\theta)\ p(\theta|\mu;\sigma^2)\ d\theta$$

where y is a Bernoulli with $p(y=1|x)=\theta$, $p(y=-1|x)=1-\theta$ and θ follows a logit normal distribution with parameters $\mu(x), \sigma^2(x)$ given by:

$$ln\left(\frac{\theta}{1-\theta}\right) \sim N\left(\mu;\sigma^2\right)$$

Therefore, the integral we are handling here has the following form:

$$\mathscr{L}\left(\mu,\sigma^2|y\right) = \int_{-\infty}^{\infty} \frac{1}{1+\mathrm{e}^{-yx}}\ \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}\ dx$$

Computation

One way to estimate the integral above efficiently is Monte-Carlo simulation, which is an unbiased way. We can do a Monte-Carlo from a standard normal distribution to calculate:

$$\mathscr{L}\left(\mu,\sigma^2|y\right) = E\left(\frac{1}{1+\mathrm{e}^{-yx}}|\ x \sim N\left[\mu;\sigma^2\right]\right)$$

Combine samples $\{(x_i,y_i)\}$ together, our cost function is:

$$\mathscr{C}\left(\mu(x),\sigma^2(x)\Big|\ y\right) = \sum_{i=1}^{n} ln\left(\int_{-\infty}^{\infty}\frac{1}{1+\mathrm{e}^{-y_i t}}\ \frac{1}{\sqrt{2\pi}\,\sigma(x_i)} exp\left\{-\frac{[t-\mu(x_i)]^2}{2\,\sigma^2(x_i)}\right\}\ dt\right)$$

In the training process, all we need is the calculation of derivative of μ and σ:

$$\frac{\partial\mathscr{C}}{\partial w_\mu} = \sum_{i=1}^{k}\frac{E\left\{\frac{1}{1+\mathrm{e}^{-y_i t}}\frac{\mu(x_i)-t}{\sigma^2(x_i)}|\ t\sim N\left[\mu(x_i),\sigma^2(x_i)\right]\right\}}{E\left\{\frac{1}{1+\mathrm{e}^{-y_i t}}|\ t\sim N\left[\mu(x_i),\sigma^2(x_i)\right]\right\}}\frac{\partial\mu(x_i)}{\partial w_\mu}$$

$$\frac{\partial\mathscr{C}}{\partial w_\sigma} = \sum_{i=1}^{k}\frac{E\left\{\frac{1}{1+\mathrm{e}^{-y_i t}}\left[\frac{\left(t-\mu(x_i)\right)^2}{\sigma^3(x_i)}-\frac{1}{\sigma(x_i)}\right]|\ t\sim N\left[\mu(x_i),\sigma^2(x_i)\right]\right\}}{E\left\{\frac{1}{1+\mathrm{e}^{-y_i t}}|\ t\sim N\left[\mu(x_i),\sigma^2(x_i)\right]\right\}}\frac{\partial\sigma(x_i)}{\partial w_\sigma}$$

here k denotes the mini-batch size.

Problems

1. Maximum likelihood estimation over $_\theta(x)$ and $\sigma^2_\theta(x)$ might overfit
2. Still suffer the expensive computational cost, and given the Monte-Carlo simulation, the training procedure may take longer than normal gradient descent with closed form derivatives.

2. Calibration Method of Probability Estimation

Given the fact that deep neural networks are not well-calibrated, we can find some calibration methods to give a better prediction on classification probability p. A proper score rule should be used here, and negative log-likelihood happens to be one. We just need a little modification on training process: insted of using classification error as criterion on validation set, we use NLL loss to determine hyperparameters.

Several criterions can be used to justify the performance of our calibration methods, as described above:

1. Reliability Diagrams
2. Expected Calibration Error (ECE)
3. Maximum Calibration Error (MCE)

Problem

- How to estimate the performance of variance estimation has yet undetermined. Especially in the case of classification, we cannot observe the probability directly.

3. Estimation Variance

Main Idea

Following (Nix and Weigend, 1994), we use a network that outputs two values in the final layer, corresponding to the predicted mean $\mu(x)$ and variance $\sigma^2(x)$. We ensemble M deep neural networks together, output $\mu_1, ..., \mu_M$, $\sigma^2_1(x), ..., \sigma^2_M(x)$. Therefore, the estimation of mean and variance is given by:

$$\hat{\mu}(x) = \frac{1}{M}\sum_{i=1}^{M} \mu_i(x)$$

$$\hat{\sigma}^2(x) = \frac{1}{M}\sum_{i=1}^{M} \sigma^2_i(x)$$

Also, we can estimate the variance of our estimation:

$$\hat{\sigma}^2_f = \frac{1}{M-1}\sum_{i=1}^{M} (\mu_i(x) - \hat{\mu}(x))^2$$

Using the estimators above, we can construct the prediction interval on test set.

Problems

- Under the random initialization, the independency of networks is undetermined.
- We may face overfitting problem here since we estimate mean and variance together.