Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles
Lakshminarayanan et al. (2017)

Xiayan Ji

Overview

Problem Statement

- NNs are poor at quantifying predictive uncertainty, and tend to produce overconfident predictions.
- The 'ground truth' uncertainty estimates are usually not available.
- Two evaluation measures:
    - *Calibration*: a frequentist notion of uncertainty which measures the discrepancy between subjective forecasts and (empirical) long-run frequencies.
    - *Generalization of predictive uncertainty under domain shift*: when a model is trained on one dataset, and evaluated on a completely different dataset, it should output high uncertainty.

Problem Statement

- NNs are poor at quantifying predictive uncertainty, and tend to produce overconfident predictions.
- The 'ground truth' uncertainty estimates are usually not available.
- Two evaluation measures:
  - *Calibration*: a frequentist notion of uncertainty which measures the discrepancy between subjective forecasts and (empirical) long-run frequencies.
  - *Generalization of predictive uncertainty under domain shift*: when a model is trained on one dataset, and evaluated on a completely different dataset, it should output high uncertainty.

Problem Statement

- NNs are poor at quantifying predictive uncertainty, and tend to produce overconfident predictions.
- The ‘ground truth’ uncertainty estimates are usually not available.
- Two evaluation measures:
  - *Calibration*: a frequentist notion of uncertainty which measures the discrepancy between subjective forecasts and (empirical) long-run frequencies.
  - *Generalization of predictive uncertainty under domain shift*: when a model is trained on one dataset, and evaluated on a completely different dataset, it should output high uncertainty.

Bayesian Formalism

- Bayesian formalism, whereby a prior distribution is specified upon the parameters of a NN. Then, given the training data, the posterior distribution over the parameters is computed, and is used to quantify predictive uncertainty.
- The quality of predictive uncertainty obtained using Bayesian NNs crucially depends on:
  1. the degree of approximation due to computational constraints;
  2. whether the prior distribution is 'correct', as priors of convenience can lead to unreasonable predictive uncertainties.

Approximation

Since exact Bayesian inference is computationally intractable for NNs, a variety of approximations have been developed:

- Laplace approximation
- Markov chain Monte Carlo (MCMC)
- Variational Bayesian methods
- Stochastic gradient MCMC variants (Langevin diffusion methods and Hamiltonian methods)

Dropout

- Interestingly, dropout may also be interpreted as ensemble model combination where the predictions are averaged over an ensemble of NNs (with parameter sharing).
- This interpretation motivates the investigation of ensembles as an alternative solution for estimating predictive uncertainty.

Why?

When and why can an ensemble of NNs be expected to produce good uncertainty estimates? To begin with, the "hypothesis class of the prior" is a set of predictors.

- *Soft model selection*: Bayesian model averaging (BMA) assumes that the true model is within the set, and performs soft model selection to find the best model within the set;
- *Model combination*: ensembles combine models to obtain a more powerful model; they can be expected to be better when the true model is not within the set.

Contribution

1. Describe a simple and scalable method for estimating predictive uncertainty estimates from NNs.
2. Propose a series of tasks for evaluating the quality of the predictive uncertainty estimates, in terms of calibration and generalization to unknown classes in supervised learning problems. Serve as a benchmark for comparing predictive uncertainty estimates obtained using different Bayesian/non-Bayesian/hybrid methods.

Novelty and Significance

- Investigate usefulness of ensembles for predictive uncertainty estimation, and compare their performance to SOTA Bayesian methods on a series of classification and regression benchmark datasets.
- Simpler to implement, requires surprisingly few modifications to standard NNs, and well suited for distributed computation, thereby making it attractive for large-scale deep learning applications.
- Spark more interest in non-Bayesian approaches for predictive uncertainty estimation.

Overview

Problem setup

Training dataset $\mathcal{D}$ with N i.i.d datapoints: $x \in \mathbb{R}^D, \mathcal{D} = \{x_n, y_n\}_{n=1}^N$.

- For classification: $y \in \{1, \ldots, K\}$.
- For regression: $y \in \mathbb{R}$.

Given the input features x, we use a neural network to model the probabilistic predictive distribution $p_\theta(y|x)$ over the labels, where θ are the parameters of the NN.

A simple recipe: (1) use a proper scoring rule as the training criterion, (2) use adversarial training (Goodfellow et al, 2015) to smooth the predictive distributions, and (3) train an ensemble.

Let M denote the number of NNs in the ensemble and $\{\theta_m\}_{m=1}^M$ denote the parameters of the ensemble.

Proper Scoring Rules

A scoring rule assigns a numerical score to a predictive distribution $p_\theta(y|x)$, rewarding better calibrated predictions over worse. (higher, better).

- Let a scoring rule be a function $S(p_\theta, (y, x))$ that evaluates the quality of the predictive distribution $p_\theta(y|x)$ relative to an event $y|x \leftarrow q(y|x)$ where $q(y, x)$ denotes the true distribution of (y, x)-tuples.
- The score: $S(p_\theta, q) = \int q(y, x) S(p_\theta, (y, x)) dydx$.
- A strictly proper scoring rule: $S(p_\theta, q) \leq S(q, q)$ with equality if and only if $p_\theta(y|x) = q(y|x), \forall p_\theta, q$.
- To encourage calibration, train model by minimizing the loss $L(\theta) = -S(p_\theta, q)$.

Proper Scoring Rules

A scoring rule assigns a numerical score to a predictive distribution $p_\theta(y|x)$, rewarding better calibrated predictions over worse. (higher, better).

- Let a scoring rule be a function $S(p_\theta, (y, x))$ that evaluates the quality of the predictive distribution $p_\theta(y|x)$ relative to an event $y|x \leftarrow q(y|x)$ where $q(y, x)$ denotes the true distribution of (y, x)-tuples.
- The score: $S(p_\theta, q) = \int q(y, x) S(p_\theta, (y, x)) dy dx$.
- A strictly proper scoring rule: $S(p_\theta, q) \leq S(q, q)$ with equality if and only if $p_\theta(y|x) = q(y|x), \forall p_\theta, q$.
- To encourage calibration, train model by minimizing the loss $L(\theta) = -S(p_\theta, q)$.

Proper Scoring Rules

A scoring rule assigns a numerical score to a predictive distribution $p_\theta(y|x)$, rewarding better calibrated predictions over worse. (higher, better).

- Let a scoring rule be a function $S(p_\theta, (y, x))$ that evaluates the quality of the predictive distribution $p_\theta(y|x)$ relative to an event $y|x \leftarrow q(y|x)$ where $q(y, x)$ denotes the true distribution of (y, x)-tuples.
- The score: $S(p_\theta, q) = \int q(y, x) S(p_\theta, (y, x)) dy dx$.
- A strictly proper scoring rule: $S(p_\theta, q) \leq S(q, q)$ with equality if and only if $p_\theta(y|x) = q(y|x), \forall p_\theta, q$.
- To encourage calibration, train model by minimizing the loss $L(\theta) = -S(p_\theta, q)$.

Proper Scoring Rules

A scoring rule assigns a numerical score to a predictive distribution $p_\theta(y|x)$, rewarding better calibrated predictions over worse. (higher, better).

- Let a scoring rule be a function $S(p_\theta, (y, x))$ that evaluates the quality of the predictive distribution $p_\theta(y|x)$ relative to an event $y|x \leftarrow q(y|x)$ where $q(y, x)$ denotes the true distribution of (y, x)-tuples.
- The score: $S(p_\theta, q) = \int q(y, x) S(p_\theta, (y, x)) dy dx$.
- A strictly proper scoring rule: $S(p_\theta, q) \leq S(q, q)$ with equality if and only if $p_\theta(y|x) = q(y|x), \forall p_\theta, q$.
- To encourage calibration, train model by minimizing the loss $L(\theta) = -S(p_\theta, q)$.

Proper Scoring Rules

It turns out many common NN loss functions are proper scoring rules.

- Maximizing likelihood: $S(p_\theta, (y, x)) = \log p_\theta(y|x)$, and this is a proper scoring rule due to the Gibbs inequality: $S(p_\theta, q) = E_{q(x)} q(y|x) \log p_\theta(y|x) \le E_{q(x)} q(y|x) \log q(y|x)$.
- K-class classification: cross entropy loss is equivalent to the log likelihood and is a proper scoring rule.
- Brier score (Brier, 1950): $L(\theta) = K^{-1} \sum_{k=1}^{K} (\delta_{k=y} - p_\theta(y = k|x))^2$, i.e., minimizing the squared error between the predictive probability of a label and one-hot encoding of the correct label.

This provides justification for the common trick of training NNs by minimizing the squared error for binary labels.

Training criterion for regression

- Using MSE to optimize the estimate of the conditional mean $\mu(x)$ does not capture predictive uncertainty.
- They use a network that outputs two values in the final layer, corresponding to the predicted mean $\mu(x)$ and variance $\sigma^2(x) > 0$.
- By treating the observed value as a sample from a (heteroscedastic) Gaussian distribution with the predicted mean and variance, they minimize the negative log-likelihood:

$$-\log p_\theta\left(y_n \mid \mathrm{x}_n\right) = \frac{\log \sigma_\theta^2(\mathrm{x})}{2} + \frac{\left(y - \mu_\theta(\mathrm{x})\right)^2}{2\sigma_\theta^2(\mathrm{x})} + \text{ constant}$$

Remark: i) MLE may overfit and one could use MAP; ii) If a Gaussian model is too restrictive, one could use (1) a more complex distribution, e.g., a mixture density network, or (2) a heavy-tailed distribution.

Training criterion for regression

- Using MSE to optimize the estimate of the conditional mean $\mu(x)$ does not capture predictive uncertainty.
- They use a network that outputs two values in the final layer, corresponding to the predicted mean $\mu(x)$ and variance $\sigma^2(x) > 0$.
- By treating the observed value as a sample from a (heteroscedastic) Gaussian distribution with the predicted mean and variance, they minimize the negative log-likelihood:

$$-\log p_\theta\left(y_n \mid \mathrm{x}_n\right) = \frac{\log \sigma_\theta^2(\mathrm{x})}{2} + \frac{\left(y - \mu_\theta(\mathrm{x})\right)^2}{2\sigma_\theta^2(\mathrm{x})} + \text{ constant}$$

Remark: i) MLE may overfit and one could use MAP; ii) If a Gaussian model is too restrictive, one could use (1) a more complex distribution, e.g., a mixture density network, or (2) a heavy-tailed distribution.

Training criterion for regression

- Using MSE to optimize the estimate of the conditional mean $\mu(x)$ does not capture predictive uncertainty.
- They use a network that outputs two values in the final layer, corresponding to the predicted mean $\mu(x)$ and variance $\sigma^2(x) > 0$.
- By treating the observed value as a sample from a (heteroscedastic) Gaussian distribution with the predicted mean and variance, they minimize the negative log-likelihood:

$$-\log p_\theta\left(y_n \mid \mathrm{x}_n\right) = \frac{\log \sigma_\theta^2(\mathrm{x})}{2} + \frac{\left(y - \mu_\theta(\mathrm{x})\right)^2}{2\sigma_\theta^2(\mathrm{x})} + \text{ constant}$$

Remark: i) MLE may overfit and one could use MAP; ii) If a Gaussian model is too restrictive, one could use (1) a more complex distribution, e.g., a mixture density network, or (2) a heavy-tailed distribution.

Adversarial training to smooth predictive distributions

- Adversarial training has been found to improve the robustness of classifiers, by increasing the likelihood of the target in an ϵ-neighborhood of the observed training examples.
- Fast gradient sign method to generate adversarial examples:

$$\mathrm{x}' = \mathrm{x} + \epsilon \,\mathrm{sign}\left(\nabla_{\mathrm{x}} \ell(\theta, \mathrm{x}, y)\right)$$

- Assuming ϵ is small enough, these adversarial examples can be used to augment the original training set by treating (x', y) as additional training examples.
- Adversarial training by definition computes a direction where the loss is high, and hence it is better than training with a random direction (which might not necessarily increase the loss) for smoothing the predictive distributions.

Adversarial training to smooth predictive distributions

- Adversarial training has been found to improve the robustness of classifiers, by increasing the likelihood of the target in an ϵ-neighborhood of the observed training examples.
- Fast gradient sign method to generate adversarial examples:

$$\mathrm{x}' = \mathrm{x} + \epsilon\,\mathrm{sign}\left(\nabla_{\mathrm{x}} \ell(\theta, \mathrm{x}, y)\right)$$

- Assuming ϵ is small enough, these adversarial examples can be used to augment the original training set by treating (x', y) as additional training examples.
- Adversarial training by definition computes a direction where the loss is high, and hence it is better than training with a random direction (which might not necessarily increase the loss) for smoothing the predictive distributions.

Adversarial training to smooth predictive distributions

- Adversarial training has been found to improve the robustness of classifiers, by increasing the likelihood of the target in an ϵ-neighborhood of the observed training examples.
- Fast gradient sign method to generate adversarial examples:

$$\mathrm{x}' = \mathrm{x} + \epsilon\,\mathrm{sign}\left(\nabla_{\mathrm{x}}\ell(\theta, \mathrm{x}, y)\right)$$

- Assuming ϵ is small enough, these adversarial examples can be used to augment the original training set by treating (x', y) as additional training examples.
- Adversarial training by definition computes a direction where the loss is high, and hence it is better than training with a random direction (which might not necessarily increase the loss) for smoothing the predictive distributions.

Adversarial training to smooth predictive distributions

- Adversarial training has been found to improve the robustness of classifiers, by increasing the likelihood of the target in an ϵ-neighborhood of the observed training examples.
- Fast gradient sign method to generate adversarial examples:

$$\mathrm{x}' = \mathrm{x} + \epsilon \, \mathrm{sign} \left(\nabla_{\mathrm{x}} \ell(\theta, \mathrm{x}, y) \right)$$

- Assuming ϵ is small enough, these adversarial examples can be used to augment the original training set by treating (x', y) as additional training examples.
- Adversarial training by definition computes a direction where the loss is high, and hence it is better than training with a random direction (which might not necessarily increase the loss) for smoothing the predictive distributions.

Ensembles: training and prediction

Decision tree is commonly used as the base learner, and there are two classes of ensembles:

- *Randomization*-based approaches such as random forests, where the ensemble members can be trained in parallel without any interaction
    - *Bagging* (bootstrap aggregation): de-correlates the predictions of the individual models, and ensures that the individual models are strong (e.g., have high accuracy). Ensemble members are trained on different bootstrap resamples of the original training set.
- *Boosting*-based approaches where the ensemble members are fit sequentially.

Remark: Intuitively, using more data for training the base learners helps reduce their bias and ensembling helps reduce the variance.

Training procedure

Algorithm 1 Pseudocode of the training procedure for our method

```
1: ▷ 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]).
2: Initialize $\theta_1, \theta_2, \ldots, \theta_M$ randomly
3: for $m = 1 : M$ do                                          ▷ train networks independently in parallel
4:   Sample data point $n_m$ randomly for each net      ▷ single $n_m$ for clarity, minibatch in practice
5:   Generate adversarial example using $\mathbf{x}'_{n_m} = \mathbf{x}_{n_m} + \epsilon \operatorname{sign}\big(\nabla_{\mathbf{x}_{n_m}} \ell(\theta_m, \mathbf{x}_{n_m}, y_{n_m})\big)$
6:   Minimize $\ell(\theta_m, \mathbf{x}_{n_m}, y_{n_m}) + \ell(\theta_m, \mathbf{x}'_{n_m}, y_{n_m})$ w.r.t. $\theta_m$      ▷ adversarial training (optional)
```

Ensemble predictions

Treat the ensemble as a uniformly-weighted mixture model and combine the predictions as:

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

For classification, this corresponds to averaging the predicted probabilities. For regression, the prediction is a mixture of Gaussian distributions:

$$M^{-1} \sum \mathcal{N}\left(\mu_{\theta_m}(\mathrm{x}), \sigma^2_{\theta_m}(\mathrm{x})\right)$$

with $\mu_*(\mathrm{x}) = M^{-1} \sum_m \mu_{\theta_m}(\mathrm{x})$ and
$\sigma^2_*(\mathrm{x}) = M^{-1} \sum_m \left(\sigma^2_{\theta_m}(\mathrm{x}) + \mu^2_{\theta_m}(\mathrm{x})\right) - \mu^2_*(\mathrm{x})$ respectively.

Overview

Evaluation metrics and experimental setup

Metrics for **R**egression and **C**lassification:

- (RC) Negative log likelihood (NLL): depends on the predictive uncertainty, and it is a proper scoring rule.
- (C) Classification accuracy and the Brier score: $BS = K^{-1} \sum_{k=1}^{K} \left(t_k^* - p\left(y = k \mid \mathrm{x}^*\right)\right)^2$ where $t_k^* = 1$ if $k = y^*$, and 0 o.w.
- (R) The root mean squared error (RMSE).

Setup:

- batch size $= 100$
- Adam optimizer
- learning rate $= 0.1$
- $\epsilon = 0.01 \times$ (the range of the training data along that particular dimension)

Regression on toy datasets

- For a qualitative evaluation: a one-dimensional regression dataset consists of 20 training examples drawn as $y = x^3 + \epsilon$ where $\epsilon \sim N(0, 32)$.
- The results show: (i) learning the variance and training using a scoring rule (NLL) leads to improved predictive uncertainty and (ii) the ensemble combination improves performance, especially as we move farther from the observed training data.

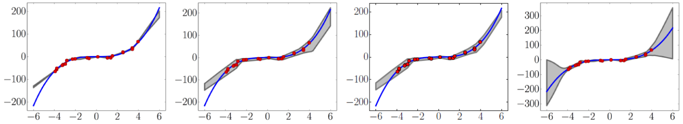
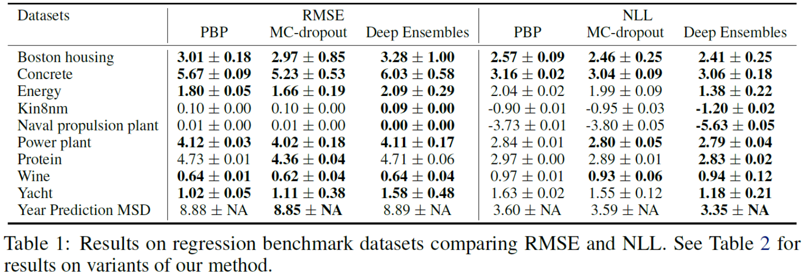


Figure 1: Results on a toy regression task: x-axis denotes x. On the y-axis, the blue line is the *ground truth* curve, the red dots are observed noisy training data points and the gray lines correspond to the predicted mean along with three standard deviations. Left most plot corresponds to empirical variance of 5 networks trained using MSE, second plot shows the effect of training using NLL using a single net, third plot shows the additional effect of adversarial training, and final plot shows the effect of using an ensemble of 5 networks respectively.

Regression on real world datasets

Outperforms (or is competitive with) existing methods in terms of NLL. On some datasets, it is slightly worse in terms of RMSE. This is because it optimizes for NLL (which captures predictive uncertainty) instead of MSE.



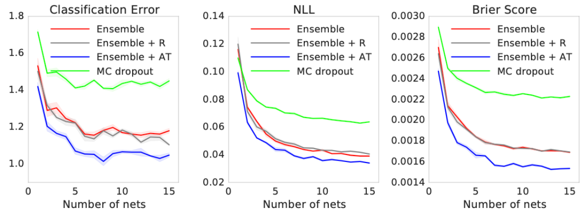
Datasets	RMSE			NLL		
	PBP	MC-dropout	Deep Ensembles	PBP	MC-dropout	Deep Ensembles
Boston housing	**3.01 ± 0.18**	**2.97 ± 0.85**	**3.28 ± 1.00**	**2.57 ± 0.09**	**2.46 ± 0.25**	**2.41 ± 0.25**
Concrete	**5.67 ± 0.09**	**5.23 ± 0.53**	**6.03 ± 0.58**	**3.16 ± 0.02**	**3.04 ± 0.09**	**3.06 ± 0.18**
Energy	**1.80 ± 0.05**	**1.66 ± 0.19**	**2.09 ± 0.29**	2.04 ± 0.02	1.99 ± 0.09	**1.38 ± 0.22**
Kin8nm	0.10 ± 0.00	0.10 ± 0.00	**0.09 ± 0.00**	-0.90 ± 0.01	-0.95 ± 0.03	**-1.20 ± 0.02**
Naval propulsion plant	0.01 ± 0.00	0.01 ± 0.00	**0.00 ± 0.00**	-3.73 ± 0.01	-3.80 ± 0.05	**-5.63 ± 0.05**
Power plant	**4.12 ± 0.03**	**4.02 ± 0.18**	**4.11 ± 0.17**	2.84 ± 0.01	**2.80 ± 0.05**	**2.79 ± 0.04**
Protein	4.73 ± 0.01	**4.36 ± 0.04**	4.71 ± 0.06	2.97 ± 0.00	2.89 ± 0.01	**2.83 ± 0.02**
Wine	**0.64 ± 0.01**	**0.62 ± 0.04**	**0.64 ± 0.04**	0.97 ± 0.01	**0.93 ± 0.06**	**0.94 ± 0.12**
Yacht	**1.02 ± 0.05**	**1.11 ± 0.38**	**1.58 ± 0.48**	1.63 ± 0.02	1.55 ± 0.12	**1.18 ± 0.21**
Year Prediction MSD	8.88 ± NA	**8.85 ± NA**	8.89 ± NA	3.60 ± NA	3.59 ± NA	**3.35 ± NA**

Table 1: Results on regression benchmark datasets comparing RMSE and NLL. See Table 2 for results on variants of our method.

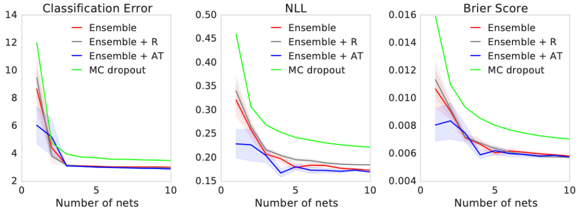
Classification on MNIST, SVHN and ImageNet

- Evaluating the effect of adversarial training as well as the number of networks in the ensemble: they significantly improve performance in terms of both classification accuracy as well as NLL and Brier score, illustrating that their method produces well-calibrated uncertainty estimates.
- Another advantage of using an ensemble is that it enables us to easily identify training examples where the individual networks disagree or agree the most. This disagreement provides another useful qualitative way to evaluate predictive uncertainty.

Cont'



(a) MNIST dataset using 3-layer MLP

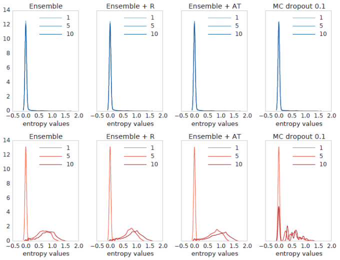


(b) SVHN using VGG-style convnet

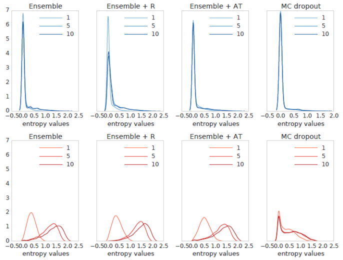
Uncertainty evaluation: test examples from known vs unknown classes

- Evaluate uncertainty on out-of-distribution examples from unseen classes, higher uncertainty when the test data is very different from the training data. Here we train on digits and test on alphabets.
- For unknown classes (bottom row), as M increases, the entropy of deep ensembles increases much faster than for MC-dropout, indicating that their method is better suited for handling unseen test examples.

Cont'd



(a) MNIST-NotMNIST

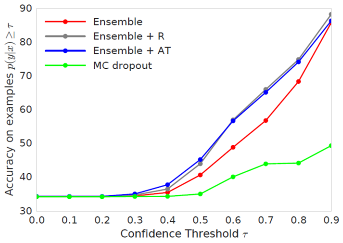


(b) SVHN-CIFAR10

Figure 3: : Histogram of the predictive entropy on test examples from known classes (top row) and unknown classes (bottom row), as we vary ensemble size M.

Accuracy as a function of confidence

Given the prediction $p(y = k \mid \mathrm{x})$, they define the predicted label as $\hat{y} = \arg\max_k p(y = k \mid \mathrm{x})$, and the confidence as $p(y = \hat{y} \mid \mathrm{x}) = \max_k p(y = k \mid \mathrm{x})$. MC-dropout can produce overconfident wrong predictions as evidenced by low accuracy even for high values of τ, whereas deep ensembles are significantly more robust.



Overview

Discussion

- Their method captures two sources of uncertainty.
  - Training a probabilistic NN $p_\theta(y|x)$ using proper scoring rules as a training objective captures ambiguity in targets y for a given x.
  - A combination of ensembles (which captures "model uncertainty" by averaging predictions over multiple models consistent with the training data), and adversarial training (which encourages local smoothness), for robustness to model misspecification and out-of-distribution examples.

Discussion

- Ensembles, even for $M = 5$, significantly improve uncertainty quality in all the cases.
- Adversarial training helps on some datasets for some metrics and is not strictly necessary in all cases.
- Requires very little hyperparameter tuning and is well suited for large scale distributed computation and can be readily implemented for a wide variety of architectures
- Non-Bayesian (yet probabilistic) approach can perform as well as Bayesian NNs.

Overview

Summary

They propose a simple and scalable non-Bayesian solution that provides a strong baseline on evaluation metrics for predictive uncertainty quantification. Future directions:

- Explicitly de-correlating networks' predictions
- Optimizing the ensemble weights, as in stacking or adaptive mixture of experts
- Implicit ensembles, where ensemble members share parameters, e.g. using multiple heads (Lee et al., 2015; Osband et al., 2016), snapshot ensembles (Huang et al., 2017) or swapout (Singh et al., 2016).

Reference

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. *Advances in neural information processing systems*, 29, 2016.