Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

Balaji Lakshminarayanan [1] **Alexander Pritzel** [1] **Charles Blundell** [1]

Abstract

Deep neural networks are powerful black box predictors that have recently achieved impressive performance on a wide spectrum of tasks. Quantifying predictive uncertainty in neural networks is a challenging and yet unsolved problem. Bayesian neural networks, which learn a distribution over weights, are currently the state-of-the-art for estimating predictive uncertainty; however these require significant modifications to the training procedure and are computationally expensive compared to standard (non-Bayesian) neural neural networks. We propose an alternative to Bayesian neural networks, that is simple to implement, readily parallelisable and yields high quality predictive uncertainty estimates. Through a series of experiments on classification and regression benchmarks, we demonstrate that our method produces well-calibrated uncertainty estimates which are as good or better than approximate Bayesian neural networks. To assess robustness to dataset shift, we evaluate the predictive uncertainty on test examples from known and unknown distributions, and show that our method is able to express higher uncertainty on unseen data. We demonstrate the scalability of our method by evaluating predictive uncertainty estimates on ImageNet.

1. Introduction

Deep learning methods such as convolutional neural networks and recurrent neural networks have achieved state-of-the-art performance on a wide variety of machine learning tasks and are becoming increasingly popular in domains such as computer vision (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), natural language processing (Mikolov et al., 2013) and bioinformatics (LeCun et al., 2015). Despite impressive classification accuracies and mean squared errors in supervised learning problems, neural networks are poor at quantifying predictive uncertainty, and tend to be overconfident in their predictions. Evaluating the quality of predictive uncertainties is challenging as the true conditional probabilities of the data are usually not available. In this work, we shall focus upon two measures of the quality of predictive uncertainty from a neural network. Firstly, we shall examine *calibration* (Dawid, 1982; DeGroot and Fienberg, 1983). Formally, calibration is the discrepancy between subjective forecasts and (empirical) long-run frequencies. This is a frequentist notion of uncertainty: if a network claims that 90% of the time a particular label is the correct label, then, during evaluation, 90% of all labels ascribed probability 90% of being correct, should be the correct label. The quality of calibration can be measured by *proper scoring rules* (Gneiting and Raftery, 2007) such as log predictive probabilities and the Brier score (Brier, 1950). Interestingly, these two proper scoring rules are commonly used in deep learning, but without reference to their properties for incentivising calibration. Note that calibration is an orthogonal concern to accuracy: a network's predictions may be accurate and yet miscalibrated. The second notion of quality of predictive uncertainty we consider concerns generalisation of the predictive uncertainty to domain shift, that is, measuring if the network *knows what it knows*. For example, if a network trained on one dataset is evaluated on a completely different dataset, then the network should output high predictive uncertainty.

Well-calibrated predictions have a number of important practical uses and lie at the heart of many forecasting problems about the physical world (e.g., weather, earthquakes, medical diagnosis, etc). Robustness to misspecification is a common requirement in many real world applications as training data is incomplete or lags behind the most recent data upon which predictions are to be made. Calibrated predictions permit modularity; if all components of a system are well-calibrated, then the uncertainty of various predictions can easily be integrated as a common currency of uncertainty between different modules.

There has been a lot of recent interest in adapting neural networks to encompass uncertainty and probabilistic methods; the majority of this work revolves around a Bayesian formalism (Bernardo and Smith, 2009), whereby a prior distribution is specified upon the weights of a single neural network and then an approximation scheme is derived that infers the posterior distribution upon the weights of the neural net-

[1] DeepMind, London, United Kingdom. Correspondence to: Balaji Lakshminarayanan <balajiln@google.com>.

work after having incorporated the training data. Approximate Bayesian approaches learn a posterior distribution over the parameters of the neural network and use this approximate posterior distribution to estimate predictive uncertainty. Early work on Bayesian neural networks focused on Markov chain Monte Carlo (MCMC) methods and Laplace approximation, cf. the seminal work of MacKay (1992) and Neal (1996). While MCMC can be used for small neural networks, it is computationally expensive for large deep neural networks. Hence, recent work on uncertainty in neural networks has focussed mostly on relatively faster approximate Bayesian solutions such as variational Bayesian methods (Blundell et al., 2015; Graves, 2011; Louizos and Welling, 2016), assumed density filtering (Hernández-Lobato and Adams, 2015), expectation propagation (Hasenclever et al., 2015; Li et al., 2015) or stochastic gradient Langevin diffusion methods (Korattikara et al., 2015; Welling and Teh, 2011). These approximations are not guaranteed to provide uncertainty estimates that reflect underlying beliefs (except, possibly, in the limit of infinite data). The quality of predictive distributions obtained using these approaches depends on (1) the degree of approximation due to computational constraints (e.g. mean field variational Bayes methods typically underestimate posterior uncertainty) and (2) *if* the prior distribution is 'correct' (e.g. the model could be misspecified); for instance, Rasmussen and Quinonero-Candela (2005) discuss an example where priors of convenience lead to unreasonable predictive uncertainties. Even the exact Bayesian posterior may not necessarily be robust to misspecification with respect to domain shift. The enormous size of the parameter space of modern neural networks compounds both of these issues. Bayesian neural networks are often computationally slower to train and harder to implement compared to their non-Bayesian counterparts, which raises the need for a 'general purpose solution' that can deliver calibrated uncertainty estimates and yet requires only minor modifications to the standard training pipeline.

Recently, Gal and Ghahramani (2016) proposed using *Monte Carlo dropout* (MC-dropout) to estimate predictive uncertainty by using *Dropout* (Srivastava et al., 2014) at test time. There has been work on approximate Bayesian interpretation (Gal and Ghahramani, 2016; Kingma et al., 2015; Maeda, 2014) of dropout. Interestingly, dropout may also be interpreted as *ensemble model combination* (Srivastava et al., 2014) (as opposed to Bayesian model averaging), where the predictions are averaged over multiple networks. The latter approximation seems more plausible particularly in the scenario where the dropout rates are not tuned based on the training data (since any sensible approximation to the true Bayesian posterior distribution has to depend on the training data). It has long been observed that ensembles of models improve overall performance (see (Dietterich, 2000) for a review). Bayesian model averaging attempts to find the single best model (or parameters) in a soft manner, assuming the true model lies within the hypothesis class of the prior. In contrast, ensembles perform *model combination*, i.e. they combine the models to obtain a more powerful model; ensembles can be expected to be better when the true model does not lie within the hypothesis class (see (Clarke, 2003; Minka, 2000) for related discussions). Hence, ensembles potentially provide a complementary source of methods for estimating predictive uncertainty.

Our contribution in this paper is two fold. First, we describe a simple, scalable method for estimating predictive uncertainty estimates from neural networks. We demonstrate that two simple modifications to the training pipeline, namely (i) *ensembles* and (ii) *adversarial training* (Goodfellow et al., 2015), are sufficient to obtain well-calibrated uncertainty estimates for supervised learning. Secondly, we propose a series of tasks for evaluating the quality of the predictive uncertainty estimates, in terms of calibration and generalisation to unknowns in supervised learning problems. These tasks, along with our simple yet strong baseline, provides a useful benchmark for comparing predictive uncertainty estimates obtained using Bayesian/non-Bayesian methods.

Ensembles of deep neural networks, or *deep ensembles* for short, have long been successfully used to boost predictive performance (e.g. classification accuracy in Imagenet or Kaggle contests) and adversarial training has been used to improve robustness to adversarial examples; however, to the best of our knowledge, ours is the first work to investigate their usefulness for predictive uncertainty estimation and compare their performance to current state-of-the-art approximate Bayesian methods on a series of classification and regression benchmark datasets. Compared to Bayesian neural networks (e.g. variational inference or Monte Carlo methods), our method is simpler to implement, well suited for distributed computation, and requires surprisingly few modifications to standard neural networks, thereby making it attractive for large-scale applications.

2. Deep ensembles for uncertainty estimation

2.1. Problem setup and High-level summary

We assume that the training dataset $\mathcal{D}$ consists of N i.i.d. data points $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, where $\mathbf{x} \in \mathbb{R}^D$ represents the D-dimensional features. For classification problems, the label is assumed to be one of K classes, that is $y \in \{1, \dots, K\}$. For regression problems, the label is assumed to be real-valued, that is $y \in \mathbb{R}$. Given a test data point $\mathbf{x}$, the goal is to output a predictive distribution $p_\theta(y|\mathbf{x})$ where θ are the parameters of the neural network.

We suggest 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 neural networks in the ensemble and $\{\theta_m\}_{m=1}^M$ denote the parameters of the ensemble. We first describe how to train a single neural net and then explain how to train an ensemble of networks.

2.2. Proper scoring rules

Scoring rules measure the quality of predictive uncertainty (see (Gneiting and Raftery, 2007) for a review). A scoring rule assigns a numerical score to a predictive distribution $p_\theta(y|\mathbf{x})$, rewarding better calibrated predictions over worse. We shall consider scoring rules where a higher numerical score is better. Let a scoring rule be a function $S(p_\theta, (y, \mathbf{x}))$ that evaluates the quality of the predictive distribution $p_\theta(y|\mathbf{x})$ relative to an event $y|\mathbf{x} \sim q(y|\mathbf{x})$ where $q(y, \mathbf{x})$ denotes the true distribution on $(y, \mathbf{x})$-tuples. The expected scoring rule is then $S(p_\theta, q) = \int q(y, \mathbf{x}) S(p_\theta, (y, \mathbf{x})) dy d\mathbf{x}$. A *proper scoring rule* is one where $S(p_\theta, q) \leq S(q, q)$ with equality if and only if $p_\theta(y|\mathbf{x}) = q(y|\mathbf{x})$, for all p_θ and q. A neural network can then be trained according to measure that encourages calibration of predictive uncertainty by minimising the loss $\mathcal{L}(\theta) = -S(p_\theta, q)$.

It turns out many common neural network loss functions are proper scoring rules. For example, when maximising likelihood, the score function is $S(p_\theta, (y, \mathbf{x})) = \log p_\theta(y|\mathbf{x})$, and this is a proper scoring rule due to Gibbs inequality: $S(p_\theta, q) = \mathbb{E}_{q(\mathbf{x})} q(y|\mathbf{x}) \log p_\theta(y|\mathbf{x}) \leq \mathbb{E}_{q(\mathbf{x})} q(y|\mathbf{x}) \log q(y|\mathbf{x})$. Interestingly, in the case of K-way classification, $S(p_\theta, (y, \mathbf{x})) = -\sum_{k=1}^K (\delta_{k=y} - p_\theta(y = k|\mathbf{x}))^2$ (i.e., minimising the squared error between the predictive probability of a label and one-hot encoding of the correct label) is also a proper scoring rule, known as the Brier score (Brier, 1950). This provides justification for this common trick for training neural networks by minimizing the square error between a binary label and its associated probability and shows it is, in fact, a well defined loss with desirable properties.[1]

2.2.1. Training criterion for regression

For regression problems, neural networks usually output a single value say $\mu(\mathbf{x})$ and the parameters are optimised to minimise the mean squared error (MSE) on the training set, given by $\sum_{n=1}^N \big(y_n - \mu(\mathbf{x}_n)\big)^2$. However, the MSE does not capture predictive uncertainty. Following (Nix and Weigend, 1994), we use a network that outputs two values in the final layer, corresponding to the predicted mean $\mu(\mathbf{x})$ and variance[2] $\sigma^2(\mathbf{x}) > 0$. By treating the observed value as a sample from a (heteroscedastic) Gaussian distribution with the predicted mean and variance, we minimise the negative log-likelihood criterion:

$$-\log p_\theta(y_n|\mathbf{x}_n) = \frac{\log \sigma_\theta^2(\mathbf{x})}{2} + \frac{\big(y - \mu_\theta(\mathbf{x})\big)^2}{2\sigma_\theta^2(\mathbf{x})} + C. \quad (1)$$

We found the above to perform satisfactorily in our experiments. However, two simple extensions are worth further investigation: (1) Maximum likelihood estimation over $\mu_\theta(\mathbf{x})$ and $\sigma_\theta^2(\mathbf{x})$ might overfit; one could impose a prior and perform maximum-a-posteriori (MAP) estimation. (2) In cases where the Gaussian is too-restrictive, one could use a complex distribution e.g. mixture density network (Bishop, 1994) or a heavy-tailed distribution.

It is tempting to use an ensemble of neural networks (trained to minimise MSE) to obtain multiple point predictions and use the empirical variance of the networks' predictions as an approximate measure of uncertainty. However, this generally does not lead to well-calibrated predictive probabilities as MSE is not a scoring rule that captures predictive uncertainty. As a motivating example, we report calibration curves (also known as reliability diagrams) on the *Year Prediction MSD* dataset in Figure 1. First, we compute the $z\%$ (e.g. 90%) prediction interval for each test data point based on Gaussian quantiles using predictive mean and variance. Next, we measure what fraction of test observations fall within this prediction interval. For a well-calibrated regressor, the observed fraction should be close to $z\%$. We compute observed fraction for $z = 10\%$ to $z = 90\%$ in increments of 10. A well-calibrated regressor should lie very close to the diagonal; on the left subplot we observe that the proposed method, which learns the predictive variance, leads to a well-calibrated regressor. However, on the right subplot, we observe that the empirical variance obtained from neural networks which do not learn the predictive variance (specifically, five neural networks trained to minimise MSE) consistently underestimates the true uncertainty. For instance, the 80% prediction interval contains only 20% of the test observations, which means the empirical variance significantly underestimates the true predictive uncertainty.

2.3. Adversarial training

Adversarial examples, proposed by Szegedy et al. (2014) and extended by Goodfellow et al. (2015), are those which are 'close' to the original training examples (e.g. an image that is visually indistinguishable from the original image to humans), but are misclassified by the neural network. Goodfellow et al. (2015) proposed the *fast gradient sign method* as a fast solution to generate adversarial examples. Given an input $\mathbf{x}$ with target y, and loss $\ell(\theta, \mathbf{x}, y)$ (e.g. $-\log p_\theta(y|\mathbf{x})$), the fast gradient sign method generates an

[1] Indeed as noted in Gneiting and Raftery (2007), it can be shown that asymptotically maximising a proper scoring rule recovers true parameter values.

[2] We enforce the positivity constraint on the variance by passing the second output through the *softplus* function $\log(1 + \exp(\cdot))$, and add a minimum variance (e.g. 10^{-6}) for numerical stability.

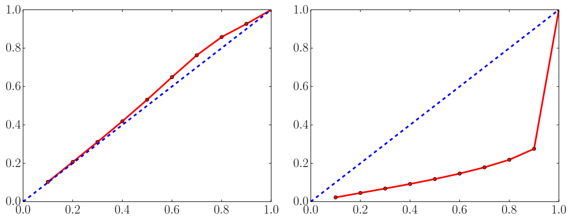

Figure 1. Calibration results on the Year Prediction MSD dataset: x-axis denotes the expected fraction and y-axis denotes the observed fraction; ideal output is the dashed blue line. Predicted variance (left) is significantly better calibrated than the empirical variance (right). See main text for further details.

adversarial example as

$$\mathbf{x}' = \mathbf{x} + \epsilon \operatorname{sign}\Big(\nabla_{\mathbf{x}}\, \ell(\theta, \mathbf{x}, y)\Big) \tag{2}$$

where ϵ is a small value such that the max-norm of the perturbation is bounded. Intuitively, the adversarial perturbation creates a new training example by adding a perturbation along a direction which the network is likely to increase the loss. Assuming ϵ is small enough, these adversarial examples can be used to augment the original training set by treating $(\mathbf{x}', y)$ as additional training examples. This procedure, referred to as *adversarial training*,[3] was found to improve the classifier's robustness.

Adversarial training can be also 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. Ideally one would want to smooth the predictive distributions along all 2^D directions in $\{1, -1\}^D$; however this is computationally expensive. A random direction might not necessarily increase the loss; however, adversarial training by definition computes the direction where the loss is high and hence is better for smoothing predictive distributions. Miyato et al. (2016) proposed a related idea called *virtual adversarial training* (VAT), where they picked $\Delta\mathbf{x} = \arg\max_{\Delta\mathbf{x}} KL\Big(p(y|\mathbf{x})||p(y|\mathbf{x} + \Delta\mathbf{x})\Big)$; the advantage of VAT is that it does not require knowledge of the true target y and hence can be applied to semi-supervised learning. Miyato et al. (2016) showed that distributional smoothing using VAT is beneficial for efficient semi-supervised learning; in contrast, we show that adversarial training is helpful for better predictive uncertainty estimation. Hence, our contributions are complementary; one could use VAT for improving predictive uncertainty in the semi-supervised setting as well.

2.4. Ensembles: training and prediction

The most popular ensembles use decision trees as the base learners and a wide variety of method have been explored in the literature on ensembles. Broadly, there are two classes of ensembles: *randomisation*-based approaches such as random forests (Breiman, 2001), where the ensemble members can be trained in parallel without any interaction, and *boosting*-based approaches where the ensemble members are fit sequentially. We focus only on the randomisation based approach as it is better suited for distributed, parallel computation. Breiman (2001) showed that the generalisation error of random forests can be upper bounded by a function of the strength and correlation between individual trees; hence it is desirable to use a *randomisation scheme* that de-correlates the predictions of the individual models as well as ensures that the individual models are strong (e.g. high accuracy). One of the popular strategies is *bagging* or bootstrapping, where ensemble members are trained on different bootstrap samples of the original training set. If the base learner lacks intrinsic randomisation (e.g. it can be trained efficiently by solving a convex optimisation problem), the bootstrap is a good mechanism for inducing diversity. However, if the underlying base learner has multiple local optima, the bootstrap is not required and can sometimes hurt performance since a base learner trained on a bootstrap sample sees only 63% unique data points. In the literature on decision tree ensembles, Breiman (2001) proposed to use a combination of *bagging* (a.k.a. bootstrapping) (Breiman, 1996) and random subset selection of features at each node. Geurts et al. (2006) later showed that the bootstrap is unnecessary if additional randomness can be injected into the random subset selection procedure. Using more data for training the base learners helps reduce their bias and ensembling helps reduce the variance.

We used the entire training dataset to train each network since deep neural networks typically perform better with more data,[4] although it is straightforward to use a random subsample if need be. We found that random initialisation of neural network parameters, along with random shuffling of the data points, was sufficient to obtain good performance. The overall training procedure is summarised in Algorithm 1.

We treat the ensemble as a uniformly-weighted mixture model and combine the predictions as

$$p(y|\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} p_{\theta_m}(y|\mathbf{x}, \theta_m). \tag{3}$$

For classification, this corresponds to averaging the pre-

[3] Not to be confused with Generative Adversarial Networks.

[4] Lee et al. (2015) independently observed that training on entire dataset with random initialization was better than bagging for deep ensembles, however their goal was to improve predictive accuracy and not predictive uncertainty.

Algorithm 1 Pseudocode of the training procedure for our method

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

dicted probabilities. For regression, the prediction is a mixture of Gaussian distributions. For ease of computing quantiles and predictive probabilities, we further approximate the ensemble prediction as a Gaussian whose mean and variance are respectively the mean and variance of the mixture.[5]

3. Experimental results

3.1. Evaluation metrics and experimental setup

For both classification and regression, we evaluate the negative log likelihood (NLL) which depends on the predictive uncertainty. NLL is a proper scoring rule and a popular metric for evaluating predictive uncertainty (Quinonero-Candela et al., 2006). For classification we additionally measure classification accuracy and the Brier score, defined as $BS = \frac{1}{K}\sum_{k=1}^{K}\big(t^*_k - p(y = k|\mathbf{x}^*)\big)^2$ where $t^*_k = 1$ if $k = y^*$, and 0 otherwise. Intuitively, our Brier score definition corresponds to creating K binary classification tasks by using a *one-hot* encoding of the true target, and measuring the average MSE between the predicted probability and the true target on the K tasks. For regression problems, we additionally measured the root mean squared error (RMSE). Unless otherwise specified, we used batch size of 100 and Adam optimiser with fixed learning rate of 0.1 in our experiments. We use the same technique for generating adversarial training examples for regression problems. Goodfellow et al. (2015) used a fixed ϵ for all dimensions; this is unsatisfying if the input dimensions have different ranges. Hence, in all of our experiments, we set ϵ to 0.01 times the range of the training data along that particular dimension.

3.2. Regression on toy datasets

First, we qualitatively evaluate the performance of the proposed method on a one-dimensional toy regression dataset. This dataset was used by Hernández-Lobato and Adams (2015), and consists of 20 training examples drawn as $y = x^3 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 3^2)$. The results are shown in Figure 2. The results clearly demonstrate that (i) learning variance leads to improved predictive uncertainty and (ii) ensemble combination improves performance, especially as we move farther from the observed training data.

In Figure 3, we consider a modified version of the toy regression task; specifically we have added a sinusoidal component to the original curve close to the origin. In absence of any other information about the underlying true function, this sinusoidal component could be either treated as noise or signal. Optimizing for MSE leads to the individual networks predicting the mean close to the sinusoid (as expected). Learning the variance leads to a qualitatively different solution where the network predicts higher uncertainty around the sinusoidal perturbation. In more complicated datasets, the preferred solution would of course depend on the number of data points and the network architecture. However this example shows that training using a scoring rule instead of MSE can lead to qualitatively different predictions.

3.3. Regression on real world datasets

In our next experiment, we compare our method to state-of-the-art methods for predictive uncertainty estimation using neural networks on regression tasks. We use the experimental setup proposed by Hernández-Lobato and Adams (2015) for evaluating PBP, which was also used by Gal and Ghahramani (2016) to evaluate MC-dropout.[6] Each dataset is split into 20 train-test folds, except for the protein dataset which uses 5 folds and the Year Prediction MSD dataset which uses a single train-test split. We use the same network architecture: 1-hidden layer neural network with ReLU nonlinearity (Nair and Hinton, 2010), containing 50 hidden units for smaller datasets and 100 hidden units for the larger protein and Year Prediction MSD datasets. We trained for 40 epochs; we refer to (Hernández-Lobato and Adams, 2015) for further details about the datasets and the experimental protocol. We used 5 networks in our ensemble. Our results are shown in Table 1, along with the PBP and MC-dropout results reported in their respective papers.

We observe that our method outperforms (or is competitive with) existing methods in terms of NLL. On some datasets, we observe that our method is slightly worse in terms of RMSE. We believe that this might be caused due to the heteroscedastic regression training criterion, which optimises for NLL instead of MSE as discussed in the toy example in

[5] The mean and variance of a mixture $\frac{1}{M}\sum \mathcal{N}\big(\mu_{\theta_m}(\mathbf{x}), \sigma^2_{\theta_m}(\mathbf{x})\big)$ are given by $\mu_*(\mathbf{x}) = \frac{1}{M}\sum_m \mu_{\theta_m}(\mathbf{x})$ and $\sigma^2_*(\mathbf{x}) = \frac{1}{M}\sum_m \big(\sigma^2_{\theta_m}(\mathbf{x}) + \mu^2_{\theta_m}(\mathbf{x})\big) - \mu^2_*(\mathbf{x})$ respectively.

[6] We do not compare to VI (Graves, 2011) as PBP and MC-dropout outperform VI on these benchmarks.

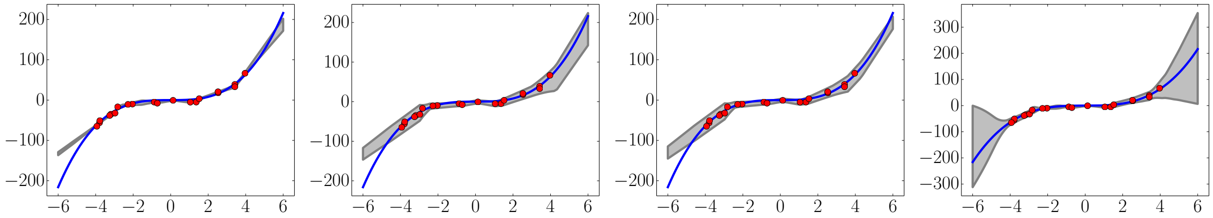

Figure 2. Results on a toy regression task: x-axis denotes x. On the y-axis, the blue line is the *ground truth* curve, the red dots are observed noisy training data points and the gray lines correspond to the predicted mean along with three standard deviations. Left most plot corresponds to empirical variance of 5 networks, second plot shows the effect of learning variance 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.

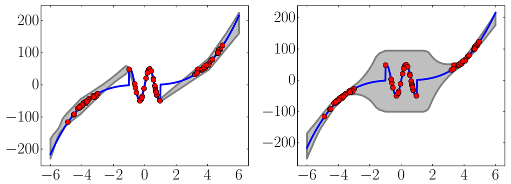

Figure 3. Results on modified toy regression task: see Figure 2 for description of x and y axes. Left plot shows empirical variance of 5 networks optimised using MSE, right plot shows variance of an ensemble of 5 networks optimised using negative log likelihood.

Datasets	RMSE			NLL		
	PBP	MCDropout	Deep Ensembles	PBP	MCDropout	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 3 for results on variants of our method.

Figure 3. Table 3 in Appendix A reports additional results on variants of our method, demonstrating the advantage of using an ensemble as well as learning variance.

3.4. Classification on MNIST, SVHN and ImageNet

Next we evaluate the performance on classification tasks using MNIST and SVHN datasets. Our goal is not to achieve the state-of-the-art performance on these problems, but rather to evaluate the effect of adversarial training as well as the number of networks in the ensemble. To verify if adversarial training helps, we also include a baseline which picks a random signed vector. For MNIST, we used an MLP with 3-hidden layers with 200 hidden units per layer and ReLU non-linearities with batch normalisation. For MC-dropout, we added dropout after each non-linearity with 0.1 as the dropout rate.[7] Results are shown in Figure 4. We observe that adversarial training and increasing the number of networks in the ensemble significantly improve performance in terms of both classification accuracy as well as NLL and Brier score, illustrating that our method produces well-calibrated uncertainty estimates. Adversarial training leads to better performance than augmenting with random direction. Our method also performs much better than MC-dropout in terms of all the performance measures. Note that augmenting the training dataset with invariances (such as random crop and horizontal flips) is complementary to adversarial training and can potentially improve performance.

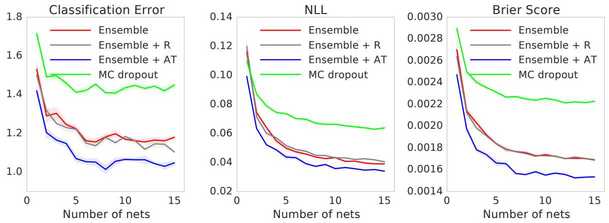

Figure 4. Results on MNIST dataset using 3-layer MLP: Both ensembles and adversarial training (AT) significantly improve performance in terms of all 3 metrics. Our method outperforms MC-dropout with the corresponding number of samples.

To measure the sensitivity of the results to the choice of network architecture, we experimented with a two-layer MLP as well as a convolutional neural network; we observed qualitatively similar results; see Appendix B in the supplementary material for details.

We also report results on the SVHN dataset using an VGG-style convolutional neural network.[8] The results are in Figure 5. Ensembles outperform MC dropout. Adversarial training helps slightly for $M = 1$, however the effect drops as the number of networks in the ensemble increases. If the classes are well-separated, adversarial training might not change the classification boundary significantly. It is not

[7] We also tried dropout rate of 0.5, but that performed worse.

[8] The architecture is similar to the one described in http://torch.ch/blog/2015/07/30/cifar.html.

clear if this is the case here, further investigation is required.

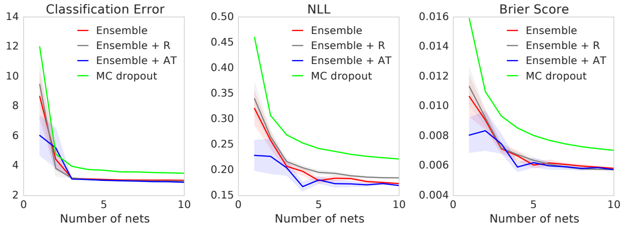

Figure 5. Results on SVHN dataset: Ensembles outperform MC-dropout on this dataset. Adversarial training helps when $M = 1$, however the effect drops as M increases.

Finally, we evaluate on the ImageNet (ILSVRC-2012) dataset (Russakovsky et al., 2015) using the *inception* network (Szegedy et al., 2016). Due to computational constraints, we only evaluate the effect of ensembles on this dataset. The results on ImageNet (single-crop evaluation) are shown in Table 2. We observe that as M increases, both the accuracy and the quality of predictive uncertainty improve significantly.

M	Top-1 error %	Top-5 error %	NLL	Brier Score $\times 10^{-3}$
1	22.166	6.129	0.959	0.317
2	20.462	5.274	0.867	0.294
3	19.709	4.955	0.836	0.286
4	19.334	4.723	0.818	0.282
5	19.104	4.637	0.809	0.280
6	18.986	4.532	0.803	0.278
7	18.860	4.485	0.797	0.277
8	18.771	4.430	0.794	0.276
9	18.728	4.373	0.791	0.276
10	18.675	4.364	0.789	0.275

Table 2. Results on ImageNet: Ensembles lead to lower classification error as well as better predictive uncertainty as evidenced by lower NLL and Brier score.

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[9] provides another useful qualitative way to evaluate predictive uncertainty. Figure 6 reports results on MNIST dataset.

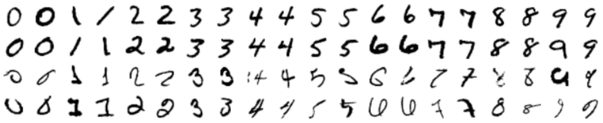

Figure 6. Results on MNIST showing test examples with high or low disagreement between the networks in the ensemble: Top two rows denote the test examples with least disagreement and the bottom two rows denote test examples with the most disagreement.

3.5. Uncertainty evaluation: test examples from known vs unknown classes

In the final experiment, we evaluate uncertainty on unseen classes. Overconfident predictions on unseen classes pose a challenge for reliable deployment of deep learning models in real world applications. We would like the predictions to exhibit higher uncertainty when the test data is very different from the training data. To test if the proposed method possesses this desirable property, we train a MLP on the standard MNIST train/test split using the same architecture as before. However, in addition to the regular test set with known classes, we also evaluate it on a test set containing unknown classes. We used the test split of the NotMNIST[10] dataset. The images in this dataset have the same size as MNIST, however the labels are alphabets instead of digits. We do not have access to the true conditional probabilities, but we expect the predictions to be closer to uniform on unseen classes compared to the known classes where the predictive probabilities should concentrate on the true targets. We evaluate the entropy of the predictive distribution and use this to evaluate the quality of the uncertainty estimates. The results are shown in Figure 7. For known classes (top row), both our method and MC-dropout have low entropy as expected. For unknown classes (bottom row), as M increases, the entropy of deep ensembles increases much faster than MC-dropout indicating that our method is better suited for handling unseen test examples. In particular, MC-dropout seems to give high confidence predictions for some of the test examples, as evidenced by the mode around 0 even for unseen classes. Comparing different variants of our method, the mode for adversarial training increases faster than the mode for vanilla ensembles indicating that adversarial training is beneficial for quantifying uncertainty on unseen classes. As an additional qualitative measure, in Figure 8, we report the examples with lowest disagreement in top two rows and highest disagreement in bottom two rows respectively. From the top two rows, we see that the ensemble agreement is highest for letter '*I*' which resembles 1 in the MNIST training dataset. From the bottom two rows, we see that the ensemble disagreement is higher for examples visually different from the MNIST training dataset.

We ran a similar experiment, training on SVHN and testing on CIFAR-10 (Krizhevsky, 2009) test set; both datasets contain $32 \times 32 \times 3$ images, however SVHN contains images of digits whereas CIFAR-10 contains images of object categories. The results are shown in Figure 9. As in the MNIST-NotMNIST experiment, we observe that MC-

[9] More precisely, we define disagreement as $\sum_{m=1}^{M} KL(p_{\theta_m}(y|\mathbf{x})||p_E(y|\mathbf{x}))$ where KL denotes the Kullback-Leibler divergence and $p_E(y|\mathbf{x}) = M^{-1}\sum_m p_{\theta_m}(y|\mathbf{x})$ is the prediction of the ensemble.

[10] Available at http://yaroslavvb.blogspot.co.uk/2011/09/notmnist-dataset.html

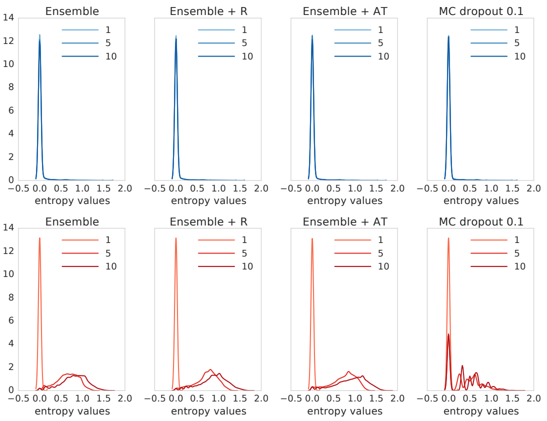

Figure 7. MNIST-NotMNIST: Histogram of the predictive entropy on test examples from known classes (top row) and unknown classes (bottom row), as we vary ensemble size M.

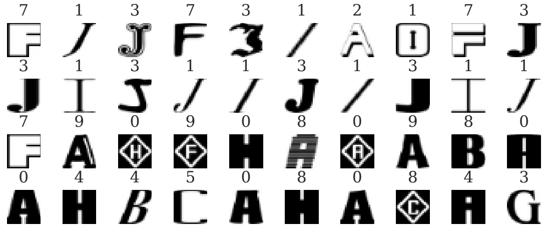

Figure 8. Network trained on MNIST and tested on the NotMNIST dataset containing unseen classes: Top two rows denote the test examples with least disagreement and the bottom two rows denote the test examples with the most disagreement.

dropout produces over-confident predictions on unseen examples, whereas our method produces higher uncertainty on unseen classes.

Finally, we test on ImageNet by splitting the training set by categories. We split the dataset into images of dogs (known classes) and non-dogs (unknown classes), following Vinyals et al. (2016) who proposed this setup for a different task. Figure 10 shows the histogram of the predictive entropy as well as the maximum predicted probability. We observe that the predictive uncertainty improves on unseen classes, as the ensemble size increases.

4. Discussion

We have proposed a simple and scalable solution that provides a very strong baseline on evaluation metrics for uncertainty quantification. Our method uses scoring rules as training objectives to encourage the neural network to produce better calibrated predictions and uses a combination of ensembles and adversarial training for robustness to model misspecification and dataset shift. Our method is well suited

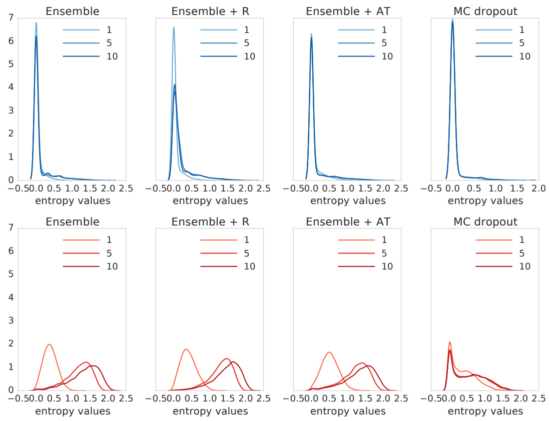

Figure 9. SVHN-CIFAR10: Histogram of the predictive entropy on test examples from known classes (top row) and unknown classes (bottom row), as we vary ensemble size M.

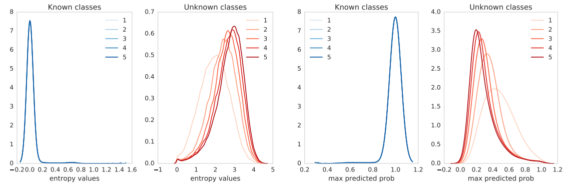

Figure 10. ImageNet trained only on dogs: Histogram of the predictive entropy (left) and maximum predicted probability (right) on test examples from known classes (dogs) and unknown classes (non-dogs), as we vary the ensemble size.

for large scale distributed computation and can be readily implemented for a wide variety of architectures such as MLPs, CNNs, etc including those which do not use dropout e.g. residual networks (He et al., 2016). It is perhaps surprising to the Bayesian deep learning community that a non-Bayesian (yet probabilistic) approach can perform as well as Bayesian NNs. We hope that this work will encourage community to think about hybrid approaches (e.g. using non-Bayesian approaches such as ensembles) and other interesting metrics for evaluating predictive uncertainty.

There are several avenues for future work. We focused on training independent networks as training can be trivially parallelised. Explicitly de-correlating networks' predictions, e.g. as in (Lee et al., 2016), might promote ensemble diversity and improve performance even further. The ensemble has M times more parameters than a single network; for memory-constrained applications, the ensemble can be distilled into a simpler model (Bucila et al., 2006; Hinton et al., 2015). It would be also interesting to investigate so-called *implicit ensembles* the 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).

ACKNOWLEDGMENTS

We would like to thank Daan Wierstra, David Silver, Ian Osband, Martin Szummer, Peter Dayan, Shakir Mohamed, Theophane Weber and Ulrich Paquet for helpful feedback.

References

J. M. Bernardo and A. F. Smith. *Bayesian Theory*, volume 405. John Wiley & Sons, 2009.

C. M. Bishop. Mixture density networks. 1994.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. In *ICML*, 2015.

L. Breiman. Bagging predictors. *Machine learning*, 24(2): 123–140, 1996.

L. Breiman. Random forests. *Machine learning*, 45(1): 5–32, 2001.

G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 1950.

C. Bucila, R. Caruana, and A. Niculescu-Mizil. Model compression. In *KDD*. ACM, 2006.

B. Clarke. Comparing Bayes model averaging and stacking when model approximation error cannot be ignored. *J. Mach. Learn. Res. (JMLR)*, 4:683–712, 2003.

A. P. Dawid. The well-calibrated Bayesian. *Journal of the American Statistical Association*, 1982.

M. H. DeGroot and S. E. Fienberg. The comparison and evaluation of forecasters. *The statistician*, 1983.

T. G. Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*. 2000.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

A. Graves. Practical variational inference for neural networks. In *NIPS*, 2011.

L. Hasenclever, S. Webb, T. Lienart, S. Vollmer, B. Lakshminarayanan, C. Blundell, and Y. W. Teh. Distributed Bayesian learning with stochastic natural-gradient expectation propagation and the posterior server. *arXiv preprint arXiv:1512.09327*, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

J. M. Hernández-Lobato and R. P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *ICML*, 2015.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29 (6):82–97, 2012.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get M for free. *ICLR submission*, 2017.

D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In *NIPS*, 2015.

A. Korattikara, V. Rathod, K. Murphy, and M. Welling. Bayesian dark knowledge. In *NIPS*, 2015.

A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra. Why M heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.

S. Lee, S. P. S. Prakash, M. Cogswell, V. Ranjan, D. Crandall, and D. Batra. Stochastic multiple choice learning for training diverse deep ensembles. In *NIPS*, 2016.

Y. Li, J. M. Hernández-Lobato, and R. E. Turner. Stochastic expectation propagation. In *NIPS*, 2015.

C. Louizos and M. Welling. Structured and efficient variational deep learning with matrix Gaussian posteriors. *arXiv preprint arXiv:1603.04733*, 2016.

D. J. MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

S.-i. Maeda. A Bayesian encourages dropout. *arXiv preprint arXiv:1412.7003*, 2014.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

T. P. Minka. Bayesian model averaging is not model combination. 2000.

T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii. Distributional smoothing by virtual adversarial examples. In *ICLR*, 2016.

V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.

R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., 1996.

D. A. Nix and A. S. Weigend. Estimating the mean and variance of the target probability distribution. In *IEEE International Conference on Neural Networks*, 1994.

I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped DQN. In *NIPS*, 2016.

J. Quinonero-Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, and B. Schölkopf. Evaluating predictive uncertainty challenge. In *Machine Learning Challenges*. Springer, 2006.

C. E. Rasmussen and J. Quinonero-Candela. Healing the relevance vector machine through augmentation. In *ICML*, 2005.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

S. Singh, D. Hoiem, and D. Forsyth. Swapout: Learning an ensemble of deep architectures. In *NIPS*, 2016.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *NIPS*, 2016.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, 2011.

Supplementary material

A. Additional results on regression benchmarks

Datasets	Ensemble-5 (MSE)	Ensemble-10 (MSE)	ML-1	ML-1 + AT	ML-5
Boston housing	3.09 ± 0.84	3.10 ± 0.83	3.17 ± 1.00	3.18 ± 0.99	3.28 ± 1.00
Concrete	5.73 ± 0.50	5.76 ± 0.50	6.08 ± 0.56	6.09 ± 0.54	6.03 ± 0.58
Energy	1.61 ± 0.19	1.62 ± 0.18	2.11 ± 0.30	2.09 ± 0.29	2.09 ± 0.29
Kin8nm	0.08 ± 0.00	0.08 ± 0.00	0.09 ± 0.00	0.09 ± 0.00	0.09 ± 0.00
Naval propulsion plant	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Power plant	4.08 ± 0.15	4.07 ± 0.15	4.10 ± 0.15	4.10 ± 0.15	4.11 ± 0.17
Protein	4.49 ± 0.04	4.50 ± 0.02	4.64 ± 0.01	4.75 ± 0.11	4.71 ± 0.17
Wine	0.64 ± 0.04	0.64 ± 0.04	0.64 ± 0.04	0.64 ± 0.04	0.64 ± 0.04
Yacht	2.78 ± 0.59	2.68 ± 0.57	1.43 ± 0.57	1.47 ± 0.58	1.58 ± 0.48
Year Prediction MSD	8.92 ± nan	8.95 ± nan	8.89 ± nan	9.02 ± nan	8.89 ± nan

Datasets	Ensemble-5 (MSE)	Ensemble-10 (MSE)	ML-1	ML-1 + AT	ML-5
Boston housing	17.28 ± 6.17	10.61 ± 4.37	2.55 ± 0.36	2.57 ± 0.37	2.41 ± 0.25
Concrete	16.07 ± 5.75	8.96 ± 1.73	3.22 ± 0.31	3.21 ± 0.26	3.06 ± 0.18
Energy	9.54 ± 4.54	6.70 ± 2.39	1.61 ± 0.40	1.51 ± 0.28	1.38 ± 0.22
Kin8nm	2.12 ± 0.97	0.11 ± 0.20	-1.11 ± 0.04	-1.12 ± 0.04	-1.20 ± 0.02
Naval propulsion plant	-5.68 ± 0.34	-5.85 ± 0.15	-5.65 ± 0.28	-4.08 ± 0.13	-5.63 ± 0.26
Power plant	35.78 ± 12.87	22.04 ± 4.42	2.82 ± 0.04	2.82 ± 0.04	2.79 ± 0.04
Protein	40.98 ± 7.43	25.73 ± 1.59	2.87 ± 0.03	2.91 ± 0.03	2.83 ± 0.02
Wine	33.73 ± 10.75	20.55 ± 3.72	1.95 ± 4.08	1.58 ± 2.30	0.94 ± 0.12
Yacht	10.18 ± 4.86	6.85 ± 2.84	1.26 ± 0.29	1.28 ± 0.36	1.18 ± 0.21
Year Prediction MSD	39.02 ± nan	21.45 ± nan	3.41 ± nan	3.39 ± nan	3.35 ± nan

Table 3. Additional results on regression benchmark datasets: the top table reports RMSE and bottom table reports NLL. Ensemble-M (MSE) denotes ensemble of M networks trained to minimise mean squared error (MSE); the predicted variance is the empirical variance of the individual networks' predictions. ML-1 denotes maximum likelihood with a single network trained to predict the mean and variance as described in Section 2.2.1 . ML-1 is significantly better than Ensemble-5 (MSE) as well as Ensemble-10 (MSE), clearly demonstrating the effect of learning variance. ML-1+AT denotes additional effect of adversarial training (AT); AT does not significantly help on these benchmarks. ML-5, referred to as *deep ensembles* in Table 1, is an ensemble of 5 networks trained to predict mean and variance. ML-5+AT results are very similar to ML-5 (the error bars overlap), hence we do not report them here.

B. Additional results on MNIST

Figures 11 and 12 report results on MNIST dataset using different architecture than those in Figure 4. We observe qualitatively similar results. Ensembles outperform MC-dropout and adversarial training improves performance.

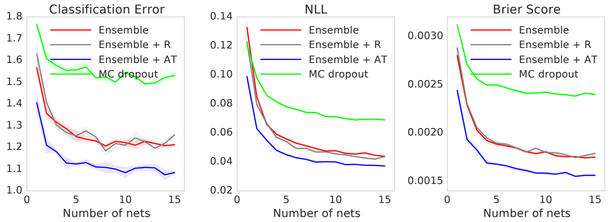

Figure 11. Results on MNIST dataset using the same setup as that in Figure 4 except that we use two hidden layers in the MLP instead of three. Ensembles and adversarial training improve performance and our method outperforms MC-dropout.

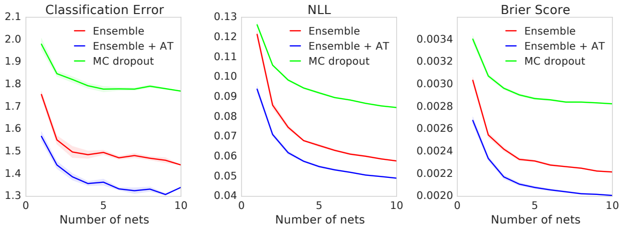

Figure 12. Results on MNIST dataset using a convolutional network as opposed to the 3-layer MLP in Figure 4. Even on a different architecture, ensembles and adversarial training improve performance and our method outperforms MC-dropout.