Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

Balaji Lakshminarayanan, Alexander Pritzel and Charles Blundell

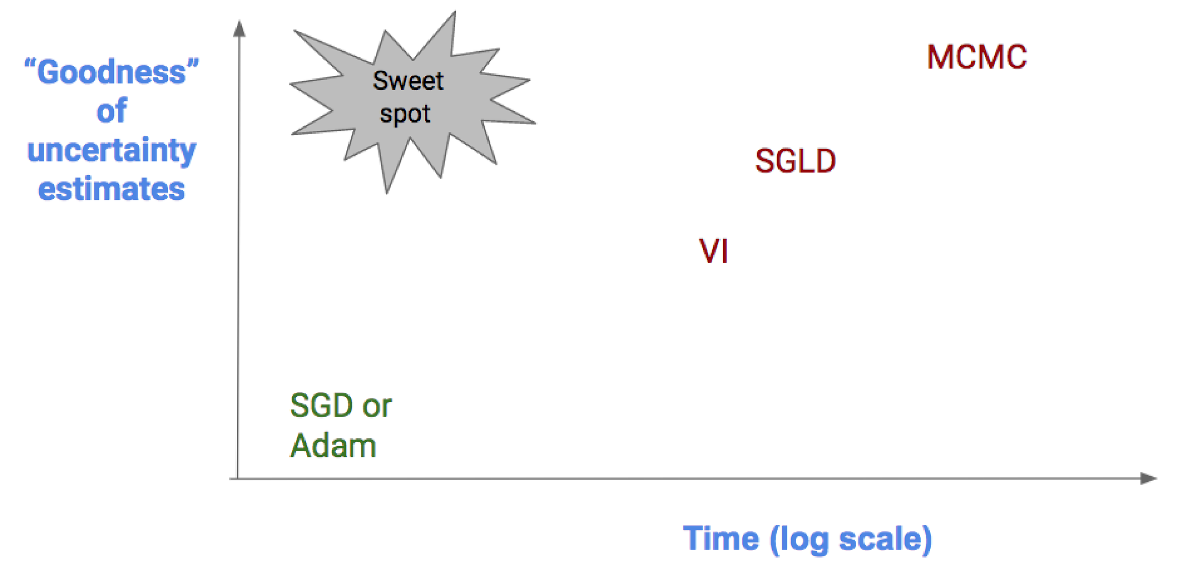
{balajiln,apritzel,cblundell}@google.com

Paper, poster, slides: **http://www.gatsby.ucl.ac.uk/~balaji**

DeepMind

Overview

- **Goal**: Predict y_* for test data x_* along with **predictive uncertainty** estimate
- **Bayesian neural networks are popular, but have some disadvantages**
  - Non-trivial modifications to training code
  - Computationally slow & difficult to scale
  - Quality of Bayesian posterior predictions depends on prior specification (model mis-specification) and posterior approximation. Bayesian approach only translates weight uncertainty to predictive uncertainty.



- **Our contributions**:
  - An experimental protocol to **measure quality of predictive uncertainty**
    - *Calibration measures* (NLL, Brier score): Frequentist coverage of subjective forecasts
    - *Robustness to dataset shift*: Is predictive uncertainty higher on test examples from unknown classes (out-of-distribution)?
  - **A simple probabilistic, non-Bayesian baseline that produces surprisingly good results**

A Simple Recipe for Uncertainty Estimation

1. Let each neural network parametrize a distribution over the outputs, i.e. $p_\theta(y|\mathbf{x})$. Use a **proper scoring rule** as training criterion
   - Classification: cross entropy loss
   - Heteroscedastic Regression: net outputs mean $\mu_\theta(\mathbf{x})$ and variance $\sigma^2_\theta(\mathbf{x})$

$$\ell(\boldsymbol{\theta}, \mathbf{x}_n, y_n) = \frac{1}{2}\log \sigma^2_\theta(\mathbf{x}) + \frac{\left(y - \mu_\theta(\mathbf{x})\right)^2}{2\sigma^2_\theta(\mathbf{x})} + \text{const.}$$

2. Augment with **adversarial training**
3. Train an **ensemble of M networks in parallel** with random initialization
4. Combine predictions at test time

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

Model combination & not Bayesian Model Averaging

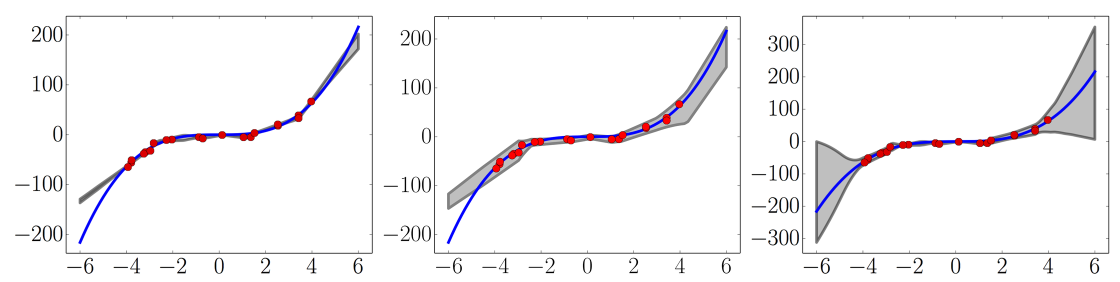
Adversarial Training

Given an input $\mathbf{x}$ with target y, create new examples $(\mathbf{x}', y)$ using the **fast gradient sign method**:

$$\mathbf{x}' = \mathbf{x} + \epsilon \ \text{sign}\Big(\nabla_{\mathbf{x}}\ \ell(\boldsymbol{\theta}, \mathbf{x}, y)\Big)$$

- Adversarial training encourages predictive distribution $p(y|\mathbf{x})$ to be similar to $p(y|\mathbf{x} + \Delta\mathbf{x})$ which encourages local smoothness and improves robustness.
- Uses gradient $\nabla_{\mathbf{x}}\ \ell(\boldsymbol{\theta}, \mathbf{x}, y)$ instead of random direction in $\Delta\mathbf{x} \in \{-1, 1\}^D$
- Can also use Virtual Adversarial training $\Delta\mathbf{x} = \arg\max_{\Delta\mathbf{x}} \mathrm{KL}\big(p(y|\mathbf{x})||p(y|\mathbf{x} + \Delta\mathbf{x})\big)$

Regression on Toy Dataset



- Blue line: **ground truth** curve, red dots: **observed noisy training data points** and gray lines: **predicted mean along with three standard deviations**
- Left plot corresponds to empirical variance of 5 networks trained using MSE, middle and right plot show the effect of learning variance using a single net and 5 networks respectively
- **Empirical variance significantly under-estimates predictive uncertainty**

Results on Regression Benchmarks

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**

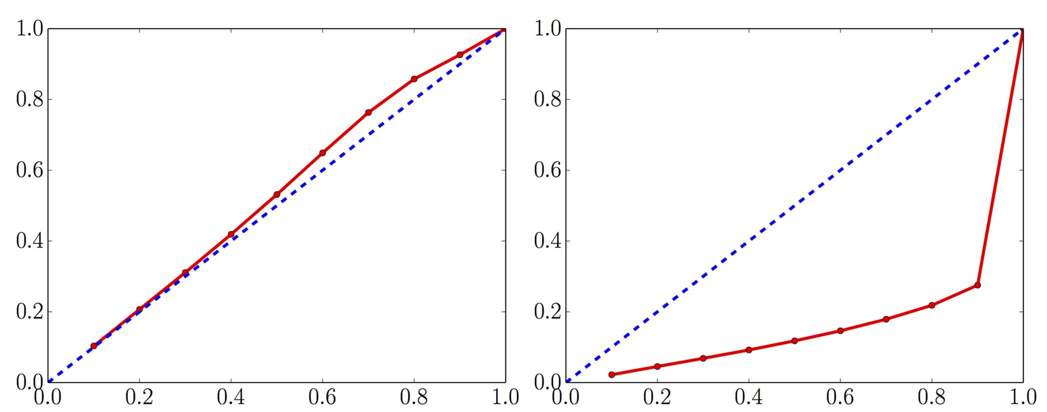


Figure: Calibration results on the Year Prediction MSD dataset: x-axis denotes the expected fraction and y-axis denotes the observed fraction; ideal output is the dashed blue line. **Predicted variance (left) is significantly better calibrated than the empirical variance (right), which is overconfident.**

Classification Results

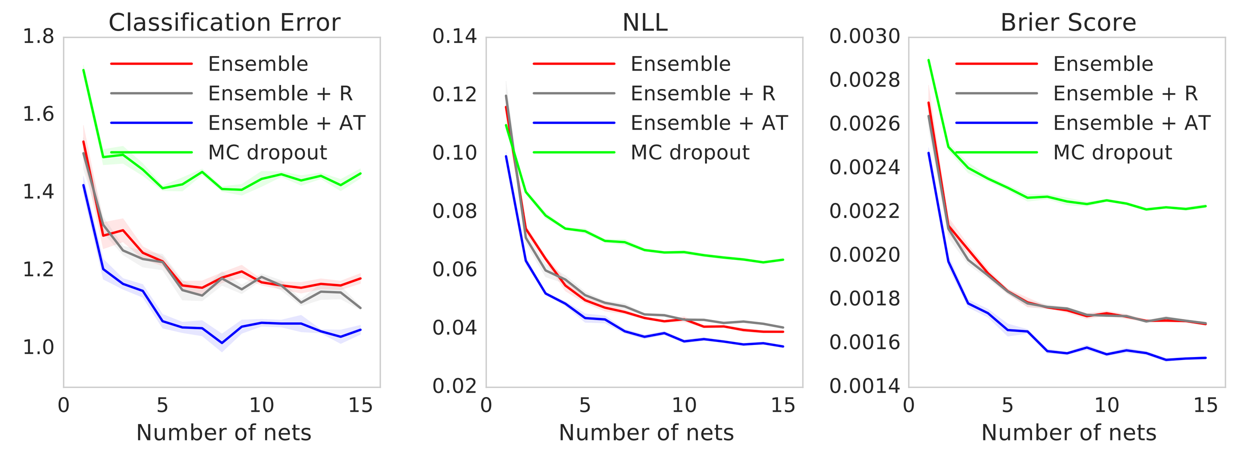


Figure: Results on MNIST using 3-layer MLP

- **Ensemble improves both classification error and predictive uncertainty (NLL, Brier score)**
- Adversarial training is better than random data augmentation
- All of our ensemble variants outperform MC-dropout
- Ensembles produce better uncertainty on other architectures and datasets as well

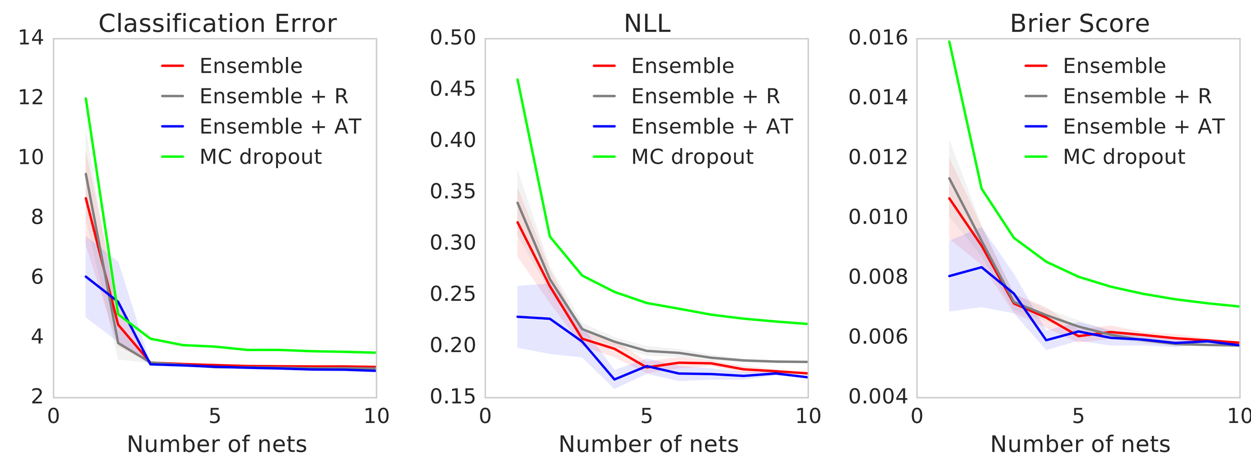


Figure: Results on on SVHN using CNNs

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: Classification on ImageNet using CNNs.

Uncertainty Evaluation on Known and Unknown Classes (Out-of-Distribution examples)

- Train MLP on standard MNIST training set. Evaluate on standard MNIST test set (known classes) as well as NotMNIST test set (unknown classes) which contains 28 × 28 images of alphabets
- **Expect higher uncertainty on unknown classes** as these inputs are far away from training data
- Measure of uncertainty: predictive entropy

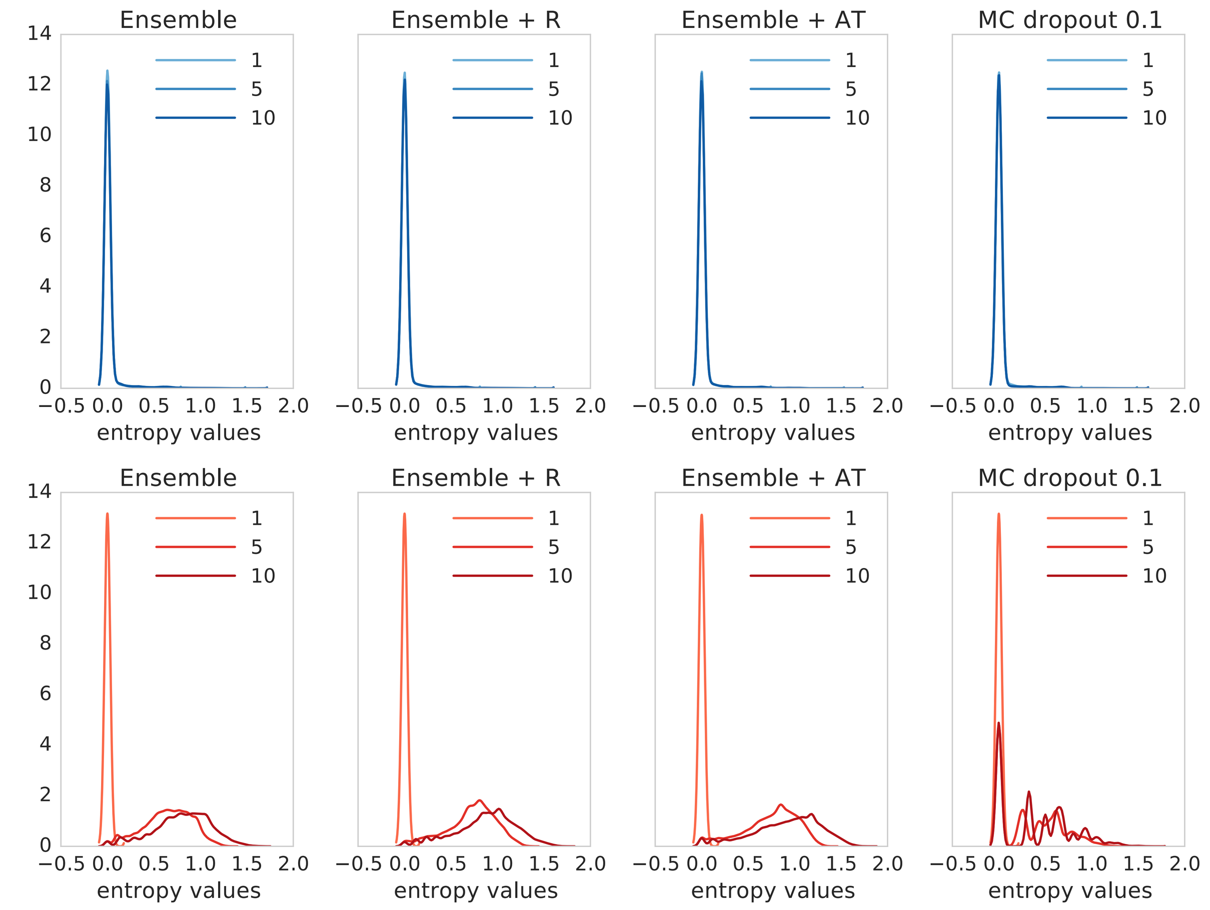


Figure: Histogram of the predictive entropy on test examples from known classes (top row) and unknown classes (bottom row), as we vary M.

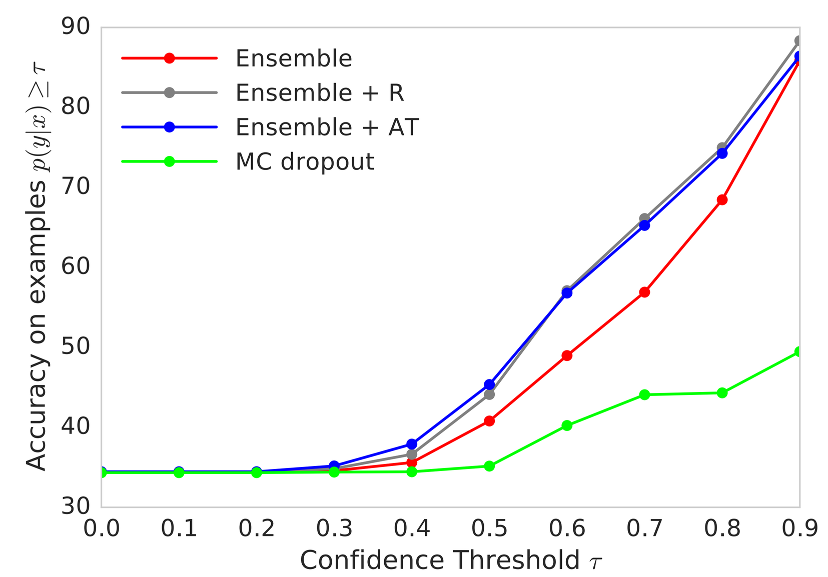


Figure: **Accuracy vs Confidence curves**: Evaluate test accuracy only on examples where $\max_y p(y|\mathbf{x}) \geq \tau$. Networks trained on MNIST and tested on both MNIST test containing known classes and the NotMNIST dataset containing unseen classes. **MC-dropout can produce overconfident wrong predictions, whereas deep ensembles are significantly more robust.**

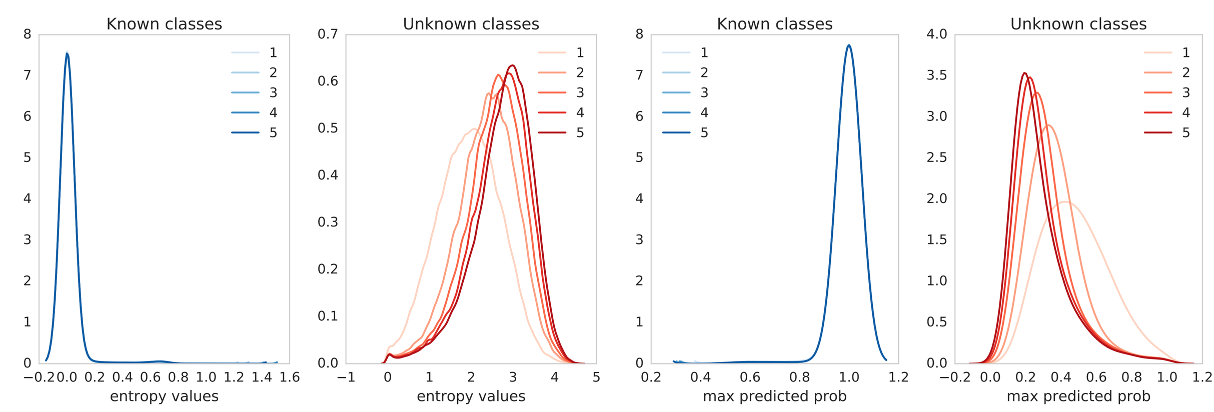


Figure: ImageNet trained only on dogs: Histogram of the predictive entropy (left) and maximum predicted probability (right) on test examples from known classes (dogs) and unknown classes (non-dogs), as we vary M.

Summary

- **Non-Bayesian method, yet produces surprisingly good predictive uncertainty estimates**
- Simple to implement. No need for hyperparameter tuning.
- Scalable & well-suited for parallel distributed computation
- Works for different output types (classification, regression) and wide variety of architectures (MLP, CNN)