Aalto University
Department of Computer Science

CS-E4875 Research Project
Uncertainty Quantification in Deep Learning

Zhiheng Qian - 765714 - zhiheng.qian@aalto.fi
Supervisor: Tianyu Cui

Jan 21st 2021

1 Introduction

Deep learning models have shown powerful ability when performing tasks such as computer vision and speech recognition in recent decade. However, applications generally produce the point prediction without any information of randomness or uncertainty. Uncertainty is commonly classified into aleatory and epistemic uncertainty. Aleatory uncertainty usually comes from randomness in the ambiguous training data, while epistemic uncertainty occurs due to the limited knowledge of data. As confidence interval can be used to quantify that uncertainty, it becomes increasingly crucial to obtain confidence level especially when the application is life critical. For example, high uncertainty generated by a weather forecasting system indicates that model might not work properly. What's more, uncertainty can be used to detect out-of-distribution (OOD) data, in which it has high value as the example is in a different distribution from training data.

Generally, there are three approaches to quantify uncertainty in deep learning models, including Bayesian methodology, ensemble and frequentist statistics. As deep learning model takes point estimates of weights when propagating through layers, Bayesian neural network computes the distribution of weights and results, then gives a good interval estimation of predictive uncertainty. However, as it calculates a distribution rather than a single point of a neuron weight, the computation of training process becomes surprisingly heavy.

In this research project, we conduct a literature review on the topic of uncertainty quantification in deep learning. Then we study and carry out a benchmark on three state-of-the-art algorithms in ensemble to perform classification and time-series prediction tasks. In the end, we summarise the learning and further investigation.

2 Literature Review

Lakshminaraynan et al. [1] proposed deep ensemble to present the idea of using an ensemble of neuron networks to get high quality uncertainty estimate. Fort and Hu [2] investigated that in deep ensemble, different solutions of same model can achieve different local minima to get better performance. Garipov et al. [4] explored the loss surface of deep learning model from the perspective of geometric feature to construct ensembles that can quantify uncertainties. Recently, Wen et al. [5] designed a computation technique which significantly speeds up the training process to form ensembles efficiently in batches. And Wenzel et al. [3] proposed hyper-deep ensemble that involves a random search for the optimal hyperparameters.

3 Methods

3.1 Deep Ensemble

Ensemble is a technique in supervised machine learning. It combines several weak supervised models to one that performs better than it's members. The proposed algorithm [1] uses ensembles of neural networks to boost predictive performance and improve scalability. Training with adversarial examples (Szegedy et al. [6]), the ensemble is robust to OOD data. During training, we use fast gradient sign method (Goodfellow et al. [7]) to generate adversarial examples:

$$x\prime = x + \epsilon sign(\bigtriangledown_x l(\theta, x, y)) \tag{1}$$

The algorithm is described below:

Algorithm 1: Training procedure for deep ensemble with adversarial examples. [1]

Result: Ensemble of size M
initialize $\theta_1, \theta_2, ..., \theta_M$ randomly ;
for $m = 1 : M$ **do**
  randomly sample training data for M_i ; `// training can be done in parallel`
  generate adversarial by equation (1) ; `// ϵ is 1% of input range`
  train with loss $l = l(\theta_m, x, y) + l(\theta_m, x\prime, y)$
end

Proper scoring rules also help with measuring the quality of predictive uncertainty. The algorithm uses proper scoring rules such as negative log-likelihood (2) and brier score (3). For regression: The common loss function such as Mean Squared Error does not reflect on predictive uncertainty. A trick[10] is to predict two values: predicted mean $\mu(x)$, *variance* $:\sigma^2 > 0$. Treating them as sample from Gaussian distribution, we can compute the negative log-likelihood in (2).

$$-logp_\theta(y_n|\mathbf{x}_n) = \frac{log\sigma_\theta^2(\mathbf{x})}{2} + \frac{(y - \mu_\theta(\mathbf{x}))^2}{2\sigma_\theta^2(\mathbf{x})} + c \tag{2}$$

$$\mathcal{L}(\theta) = -S(p\theta, (y, \mathbf{x})) = K^{-1}\Sigma_{k=1}^{K}(\delta_{k=y} - p_\theta(y = k|\mathbf{x}))^2 \tag{3}$$

3.2 BatchEnsemble

As the cost of training and testing ensemble grows linearly when the number of members increases, it easily becomes difficult and expensive to obtain. BatchEnsemble can accelerate

the training speed and memory usage by sharing one heavy weight for all members while having trainable fast weights in each ensemble member. It achieves 3X speedup at test time and 3X times less memory usage by an ensemble of 4.

BatchEnsemble propagates one heavy weight across the training. For each member, it has a tuple of trainable vectors r_i and s_i (fast weights) with the same size of input and output channels, where i indicates the index of ensemble. To get the weight of the layer, we multiply the heavy with the rank-one matrix of two light weights[5]:

$$\overline{W_i} = W \circ F_i, \text{where } F_i = r_i s_i^T \tag{4}$$

To simplify the computation, BatchEnsemble introduces vectorization to calculate the weights of each member. Let x denote the activation of the previous layer, current layer's weight are now:

$$\begin{aligned} y_n = \ & \phi(\overline{W}_i^T x_n) \\ & \phi((W^T(x_n \circ r_i)) \circ s_i) \\ & \phi(((X \circ R)W) \circ S) \end{aligned} \tag{5}$$

During testing, the fast weights are combined then averaged from the predictions of every member. Then repeating the input batch size for M times to enable each member produce the output of the same batch size in a single forward pass.

3.3 Fast Geometric Ensembling

Snapshot ensembles (Huang, Li et al. [8]) uses stochastic gradient descent with warm restarts to adjust the learning rate during training. This adjustment enables caching a snapshot of the weights as the model converges in a local minima. Repeat that by resetting the learning rate to a higher value, we obtain an ensemble of neural networks.

Fast Geometric Ensembling (FGEnsemble) shares similar idea to produce the ensemble. During training, however, FGEnsemble uses linear piecewise cyclical learning rate schedule, while Snapshot ensemble uses cosine annealing. By using large and small batch size for training, the model can converge into flat or sharp local minima. As discovered, training and testing dataset can produce similar loss surface. With shifted test surface, the flat minima generalises better because it will not change the loss value dramatically, compared with sharp minima. (Keskar et al. [9]) For a wide minima, the proposed training method can find near-constant accuracy polygonal chains between optima. Hence, FGEnsemble only need 2 to 4 epochs in each cycle to produce ensemble members, which is considerably faster to train.

Besides that, it's also found that the optima of loss functions are connected by simple curves[4], where there is constant training and test accuracy. Two parametrizations are proposed for the curves, including Polygonal chain (6) and Bezier curve (7):

$$\phi_\theta(t) = \begin{cases} 2(t\theta + (0.5 - t)\check{w}_1), & 0 \le t \le 0.5 \\ 2((t - 0.5)\check{w}_2 + (1 - t)\theta), & 0.5 \le t \le 1 \end{cases} \tag{6}$$

$$\phi_\theta(t) = (1-t)^2\check{w}_1 + 2t(1-t)\theta + t^2\check{w}_2, 0 \le t \le 1 \tag{7}$$

4 Experiments

In the first experiment*, we compare the performance of Deep Ensemble (with and without adversarial training) and BatchEnsemble, respectively on CIFAR-10 and MIMIC-III datasets. We use the identical network architecture for both algorithms. For both models, we evaluate the negative log likelihood to help capture the predictive uncertainty during training. We additionally calculate the brier score to measure the training accuracy. In the image classification task, we implement Deep Ensemble with ensemble size $M = 4$, each member is a Wide ResNet 28-4 (WRN28-4) model trained with NLL. With learning rate of 0.01 and dropout rate of 0.5, we observe that training converges around 40 epochs. To implement BatchEnsemble, we use customized convolutional layer and fully-connected layer and propagate weights using (5). In the sequence processing task, we implement a recurrent neural network a sequential container of one input dense layer, lstm layer with 64 hidden units and one output dense layer. We choose Adam optimizer with 0.001 learning rate the 0.0004 weight decay for the training, it shows convergence for 20 epochs.

		Deep Ensemble	Deep Ensemble (adversarial)	BatchEnsemble
WRN28-4	Training time	$1639.859 \pm 11.602s$	$3001.936 \pm 59.331s$	$1680.635 \pm 29.846s$
	NLL loss	0.064 ± 0.044	0.339 ± 0.042	0.130 ± 0.067
	Brier score	0.033 ± 0.033	0.004 ± 0.000	0.005 ± 0.001
	Test Accuracy	0.825 ± 0.010	0.844 ± 0.005	0.852 ± 0.006
	Test Accuracy (adversarial)	0.826 ± 0.008	0.844 ± 0.005	NA
LSTM	Training time	$406.204 \pm 1.704s$	$1055.907 \pm 8.963s$	$1210.124 \pm 8.624s$
	NLL loss	0.241 ± 0.001	0.324 ± 0.009	0.341 ± 0.000
	Brier score	0.003 ± 0.000	0.003 ± 0.000	0.001 ± 0.000
	Test Accuracy	0.867 ± 0.260	0.869 ± 0.230	0.871 ± 0.128
	Test Accuracy (adversarial)	0.781 ± 0.012	0.791 ± 0.020	NA

Table 1: Comparison between Deep Ensemble and BatchEnsemble. Adversarial training is introduced for Deep Ensemble to explore whether it improves the robustness of the model. The table compares them in terms of training time, negative log-likelihood loss, brier score and test accuracy. BatchEnsemble converges faster by taking the advantage of vectorization computation.

As the result shown in Table 1, adversarial example helps train the model for a better accuracy, it also improves the robustness of the model as the test accuracy on adversarial testing example is higher. However, adding adversarial examples makes computation more expensive and slow down the training process. In addition, BatchEnsemble achieves better performance on test accuracy. While we know that Deep Ensemble needs considerable memory usage when training in parallel, it suggests that BatchEnsemble has significant improvements.

We then evaluate three algorithms in terms of various size of ensemble, and compare their training time and test accuracy. We consider using same Wide ResNet 28-4 architecture

*All experiments in this project use Google Colaboratory to train models. WRN28-4 is trained with Colab GPU. LSTM is trained with Colab CPU. Implementation details available at `https://github.com/zhiheng-qian/cs4875-research-project`

for all three algorithms[†] for comparison. We train BatchEnsemble using dynamic batch size that depends on the number of models. During training, we make sure each model gets 32 batch size for calculating weights. Surprisingly, it is unstable at training time and requires different epochs to converge. Table 2 in Appendix A shows training measure on that. As a result, we use fixed batch size $B = 128$ for the training. For FGEnsemble, we first train WRN28-4 with CIFAR-10 for 40 epochs to converge, then use proposed procedure to form the ensemble. Figure 1 shows the training time and test accuracy of three algorithms.

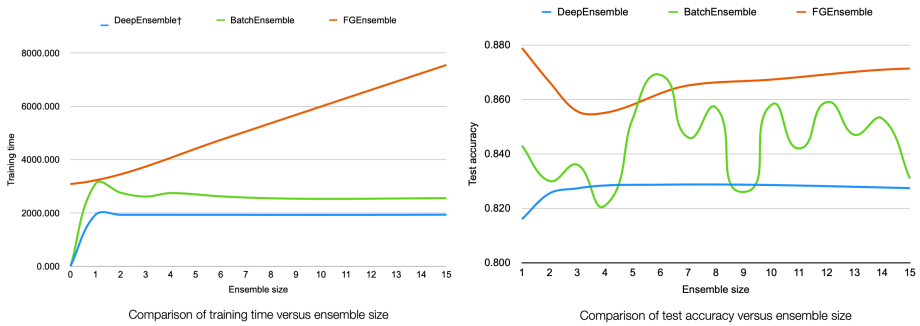

Figure 1: **Left:** Training time of each algorithm. Deep Ensemble can be trained in parallel, meanwhile have more total weight parameters as ensemble size grows. BatchEnsemble is trained slowly for 1 ensemble member, but gradually gets faster when there are more members. FGEnsemble requires pre-trained model to find endpoints, then uses proposed learning rate scheduling to form ensembles, so that the training time raises. **Right:** For test accuracy, FGEnsemble outperforms the other two ensembles. BatchEnsemble's performance is unpredictable, potential reason is that fixed batch size for all different size of ensemble may bring instability, further investigation is required.

5 Discussion

In this project we have studied and implemented three algorithms in ensemble to estimate uncertainty in Deep Learning. Deep Ensemble simply proposes the idea of producing predictive uncertainty using non-Bayesian approach. It also shows a training technique that uses adversarial examples to improve the local smoothness. What's more, proper scoring rules help with capturing ambiguity in predictions. While having all above benefits, it is the simplest and the most scalable approach. However, the memory cost grows linearly as the ensemble size increases, which makes it difficult to achieve given memory constraint.

BatchEnsemble optimizes the memory usage by sharing model parameters among all members in an ensemble, which makes the training procedure more efficient. Besides that, it can be adapted to lifelong learning by having each ensemble member responsible for one of the lifelong learning tasks.

FGEnsemble proposed a training procedure to find the pathways connecting different optima. It also explores the loss surface of deep neural networks in a fresh geometric perspective,

[†]Deep Ensemble is a scalable algorithm so each member can be trained in parallel.

which inspire a new research field that provide a new direction for Bayesian inference. And interestingly, it gives a new type of visualizations for showing the connectivity of modes.

Other learnings. Besides the algorithms studied above, it is also interesting to conduct research in such field that requires training and testing models. During the experiments, I learned that it's important to know what types of data and figures are needed, then start the training and testing since it's a quite long process. And more importantly, it's necessary to save models and checkpoints once the model improves its performance.

References

[1] Balaji Lakshminarayanan, Alexander Pritzel and Charles Blundell, Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. arXiv:1612.01474.

[2] Stanislav Fort, Huiyi Hu, Balaji Lakshminarayanan, Deep Ensembles: A Loss Landscape Perspective, arXiv:1912.02757

[3] Florian Wenzel, Jasper Snoek, Dustin Tran, Rodolphe Jenatton, Hyperparameter Ensembles for Robustness and Uncertainty Quantification, arXiv:2006.13570

[4] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov and Andrew Gordon Wilson, Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs, arXiv:1802.10026.

[5] Yeming Wen, Dustin Tran & Jimmy Ba, BATCHENSEMBLE: AN ALTERNATIVE APPROACH TO EFFICIENT ENSEMBLE AND LIFELONG LEARNING, arXiv:2002.06715.

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

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

[8] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. International Conference on Learning Representa- tions, 2017.

[9] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. International Conference on Learning Representations, 2017.

[10] Estimating the mean and variance of the target probability distribution, David A. Nix and Andreas S. Weigend. IEEE 0-7803-1901-X

A Additional results on BatchEnsemble

# of models	Batch size	Training time	NLL loss	Brier score	Test Accuracy
1	32	3099.430	0.177	0.033	0.843
2	64	2760.844	0.177	0.012	0.830
3	96	2615.517	0.156	0.007	0.836
4	128	2744.866	0.030	0.005	0.821
5	160	2626.809	0.124	0.004	0.853
6	192	2623.477	0.096	0.003	0.869
7	224	2523.506	0.099	0.003	0.846
8	256	2568.732	0.082	0.002	0.857
9	256	2569.075	0.139	0.002	0.826
10	256	2567.768	0.072	0.002	0.858
11	256	2558.224	0.173	0.002	0.842
12	256	2554.686	0.099	0.002	0.859
13	256	2554.250	0.036	0.002	0.847
14	256	2553.579	0.094	0.002	0.853
15	256	2555.975	0.265	0.002	0.831

Table 2: Training result of BatchEnsemble using dynamic batch size. As the table shows the test accuracy is unstable when the number of models in ensemble changes. What's more, we are uncertain about the sensitivity of the training process and performance for a single model to the choice of batch size. Specific experiment is needed to measure such metric.