Aalto University
Department of Computer Science

CS-E4875 Research Project
Uncertainty Quantification in Deep Learning

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

Jan 10th 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. While confidence interval can be used to quantify that uncertainty. It becomes increasingly crucial to obtain confidence level especially when the application is life critical. 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.

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.

Generally, there are three approaches to quantify uncertainty in deep learning models, including Bayesian methodology, ensembling 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.

Lakshminaraynan et al. [1] proposed deep ensemble to present the idea of using an ensemble of neuron networks to get high quality uncertainty estimate. Garipov et al. [2] 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. [3] designed a computation technique which significantly speeds up the training process to form ensembles efficiently in batches.

2 Experiments and Results

2.1 Deep Ensemble

Ensembling 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 uses ensembles of neural networks to boost predictive performance and improve scalability. Training with adversarial examples(Szegedy et al. [4]), the ensemble is robust to out-of-distribution(OOD) data. We use fast gradient sign method(Goodfellow et al. [5]) to generate adversarial examples:

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

Proper scoring rules also help with measuring the quality of predictive uncertainty. Many loss functions such as negative log-likelihood (2) and brier score (3) are proper scoring rules. For regression: The common loss function such as Mean Squared Error does not reflect on predictive uncertainty. A trick 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).

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

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

In this experiment*, we train an ensemble of 4 Wide ResNet 28-4 (WRN28-4) models with and without adversarial example from CIFAR-10. With learning rate of 0.01 and dropout rate of 0.5, we observe that training converges around 40 epochs. Then we evaluate and compare the performance on testing data and adversarial testing data. The algorithm is summarised in Algorithm 1.

Algorithm 1: Training procedure for deep ensemble with adversarial examples

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

*All experiments in this project use Google Colaboratory GPU to train models. Implementation details available at `https://github.com/zhiheng-qian/cs4875-research-project`

		Adversarial training	
		Yes	No
WRN28-4	Training time	20413.922s	8400.548s
	Training NLL loss	0.225	0.191
	Training Brier score	0.003	0.014
	Test Accuracy	0.838	0.832
	Test Accuracy(Adversarial example)	0.839	0.831

Table 1: Training Deep Ensemble with and without adversarial examples. As the table shows, adversarial example helps train the model for a better accuracy, it also improves the robustness of the model. However, adding adversarial examples makes computation more expensive and slow down the training process.

2.2 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:

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

To simplify the computation, vectorization is introduced 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, we ensemble all the fast weights and take the average of predictions from each member. To enable each member produce the output of the same batch size in a single forward pass, we repeat the input batch size for M times.

In our experiment, we use the proposed algorithm to train an ensemble of size 4 with Wide ResNet 28-4 and CIFAR-10, also with Gated Recurrent Unit(GRU) and MIMIC-III dataset. For WRN28-4, Training uses Adam optimizer with fixed learning rate 0.01, we observe that models converge around 40 epochs. Then compare the training and testing time, accuracy and loss with Deep Ensemble trained without adversarial examples.

		WRN28-4	GRU
DeepEnsemble	Training time	8400.548s	
	NLL loss	0.191	
	Brier score	0.014	
	Accuracy	0.832	
BatchEnsemble	Training time	1709.595s	
	NLL loss	0.229	
	Brier score	0.00578	
	Accuracy	0.839	

Table 2: Comparison of Deep Ensemble and Batch Ensemble in terms of training time, negative log-likelihood loss, brier score and test accuracy. Respectively on WRN28-4 with CIFAR-10, and GRU with MIMIC-III.

2.3 Fast Geometric Ensembling

Snapshot ensembles (Huang, Li et al. [6]) 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. [7]) 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, where the training and test accuracy are almost constant. Two parametrizations are proposed for the curves, including Polygonal chain (6) and Bezier curve (7):

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

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

We consider using same Wide ResNet 28-4 architecture for all three algorithms[†] for comparison. 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.

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

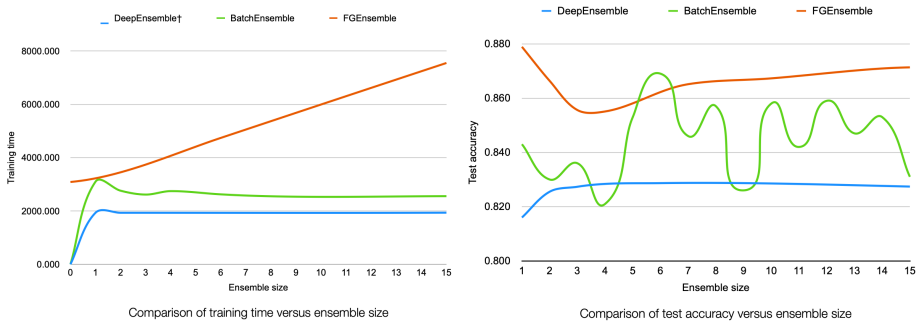

Figure 1: **Left:** Training time of each algorithm. Deep Ensemble can be trained in parallel, meanwhile have more total weight parameters as ensemble size grows. Batch Ensemble is trained slowly for 1 ensemble member, but gradually gets faster when there are more members. FGEnsemble requires pre-trained 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. Batch Ensemble's performance is unpredictable, potential reason is that fixed batch size for all different size of ensemble may bring instability, further investigation is required.

3 Discussion

In this project we have studied and implemented three algorithms in ensembling 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, 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] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov and Andrew Gordon Wilson, Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs, arXiv:1802.10026.

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

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

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

[6] 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.

[7] 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.