BatchEnsemble: Efficient Ensemble of Deep Neural Networks via Rank-1 Perturbation

Yeming Wen[1,2,3*], **Dustin Tran**[3], **Jimmy Ba**[1,2]
[1]University of Toronto, [2]Vector Institute, [3]Google Research, Brain Team

Abstract

Ensembles, where multiple neural networks are trained individually and their predictions are averaged, have been shown to be widely successful for improving both the accuracy and predictive uncertainty of single neural networks. However, an ensemble's cost for both training and testing increases linearly with the number of networks.

In this paper, we propose BatchEnsemble, an ensemble method whose computational and memory costs are significantly lower than typical ensembles. BatchEnsemble achieves this by defining each weight matrix to be the Hadamard product of a shared weight among all ensemble members and a rank-one matrix per member. Unlike ensembles, BatchEnsemble is not only parallelizable across devices, where one device trains one member, but also parallelizable within a device, where multiple ensemble members are updated simultaneously for a given mini-batch. BatchEnsemble yields competitive accuracy and uncertainties as typical ensembles; the speedup at test time is 3X and memory reduction is 3X at an ensemble of size 4.

1 Introduction

Ensembling is one of the oldest tricks in machine learning literature [Hansen and Salamon, 1990]. By combining the outputs of several models, an ensemble can achieve better performance than any of its members. Many researchers demonstrate that a good ensemble is one where the ensemble's members are both accurate and make independent errors [Perrone and Cooper, 1992, Maclin and Opitz, 1999]. In neural networks, SGD [Bottou, 2003] and its variants [Kingma and Ba, 2014] are the most common optimization algorithm. The random noise from sampling mini-batches of data in SGD-like algorithms and random initialization of the deep neural networks, combined with the fact that there is a wide variety of local minima solutions in high dimensional optimization problem [Kawaguchi, 2016, Ge et al., 2015], results in the following observation: deep neural networks trained with different random seeds can converge to very different local minima although they share similar error rates. One of the consequence is that neural networks trained with different random seeds will usually not make all the same errors on the test set, i.e. they may disagree on a prediction given the same input even if the model has converged.

Ensembles of neural networks benefit from the above observation to achieve better performance by averaging or majority voting on the output of each ensemble member [Xie et al., 2013, Huang et al., 2017]. It is shown that ensembles of models perform at least as well as its individual members and diverse ensemble members lead to better performance [Krogh and Vedelsby, 1995]. More recently, Lakshminarayanan et al. [2017] showed that deep ensembles give reliable predictive uncertainty estimates while remaining simple and scalable. A further study confirms that deep ensembles generally achieves the best performance on out-of-distribution uncertainty benchmarks [Ovadia et al., 2019] compared to other methods such as MC-dropout [Gal and Ghahramani, 2015].

4th workshop on Bayesian Deep Learning (NeurIPS 2019), Vancouver, Canada.

[*]Partial work done as part of the Google Student Researcher Program. Email: `ywen@cs.toronto.edu`

Despite their success on benchmarks, ensembles in practice are limited due to their expensive computational and memory costs, which increase linearly with the ensemble size in both training and testing. Computation-wise, each ensemble member requires a separate neural network forward pass of its inputs. Memory-wise, each ensemble member requires an independent copy of neural network weights, each up to millions (sometimes billions) of parameters. This memory requirement also makes many tasks beyond supervised learning prohibitive. For example, in lifelong learning, a natural idea is to use a separate ensemble member for each task, adaptively growing the total number of parameters by creating a new independent set of weights for each new task. No previous work achieves competitive performance on lifelong learning via ensemble methods, as memory is a major bottleneck.

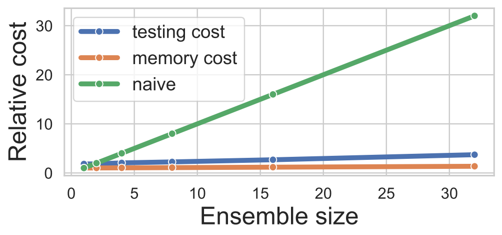


Figure 1: The test time cost (blue) and memory cost of BatchEnsemble (orange) w.r.t the ensemble size. The result is relative to single model cost. Testing time cost and memory cost of naive ensemble are plotted in green.

Our contribution: In this paper, we aim to address the computational and memory bottleneck by building a more parameter efficient ensemble model: BatchEnsemble. We achieve this goal by exploiting a novel ensemble weight generation mechanism: the weight of each ensemble member is generated by the Hadamard product between: **a.** one shared weight among all ensemble members. **b.** one rank-one matrix that varies among all members, which we refer to as fast weight in the following sections. Figure 1 compares testing and memory cost between BatchEnsemble and naive ensemble. Unlike typical ensembles, BatchEnsemble is mini-batch friendly, where it is not only parallelizable across devices like typical ensembles but also parallelizable within a device. Moreover, it incurs only minor memory overhead because a large number of weights are shared across ensemble members.

Empirically, we show that BatchEnsemble has the best trade-off among accuracy, running time, and memory in terms of calibrated prediction on CIFAR-10 corruption dataset[Hendrycks and Dietterich, 2019]. BatchEnsemble also achieves comparable uncertainty evaluation to naive ensembles on contextual bandits task.

Ensembles for Improved Uncertainty: Although deep neural networks achieve state-of-the-art performance on a variety of benchmarks, their predictions are often poorly calibrated. Bayesian neural networks [Hinton and Neal, 1995], which fit a distribution to the weights rather than a point estimate, are often used to model uncertainty. However, they requires modifications to the traditional neural network training scheme. Deep ensembles have been proposed as a simple and scalable alternative, and have been shown to make well-calibrated uncertainty estimates [Lakshminarayanan et al., 2017]. Several metrics had been proposed to measure the quality of uncertainty estimates. In section 3.1, we use Expected Calibrated Error (ECE) [Guo et al., 2017, Naeini et al., 2015] as an uncertainty metric to evaluate the performance of BatchEnsemble and other baseline methods. Section 3.2 presents the results on contextual bandits benchmark [Riquelme et al., 2018], where maximizing reward is of direct interest; this requires good uncertainty estimates in order to balance exploration and exploitation.

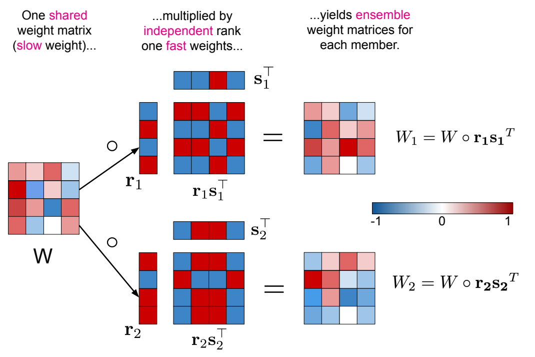
2 Methods

2.1 BatchEnsemble

In this section, we introduce how to ensemble neural networks in an efficient way. Let W be the weights in a neural network layer. Denote the input dimension as m and the output dimension as n, i.e. $W \in \mathcal{R}^{n\times m}$. For ensemble, assuming the ensemble size is M and each ensemble member has weight matrix $\overline{W}_i$. Each ensemble member owns a tuple of trainable vectors r_i and s_i which share the same dimension as output and input (n and m) respectively, where i ranges from 1 to M. Our algorithm generates a family of ensemble weights $\overline{W}_i$ by the following:

$$\overline{W}_i = W \circ F_i, \text{ where } F_i = r_i s_i^\top, \tag{1}$$

For each training example in the mini-batch, it receives an ensemble weight $\overline{W}_i$ by element-wise multiplying W, which we refer to as "slow weights", with a rank-one matrix F_i, which we refer to as "fast weights." The subscript i represents the selection of ensemble member. Since W is shared across ensemble members, we term it as "shared weight" in the following paper. Figure 2 visualizes BatchEnsemble.


One shared weight matrix (slow weight)...
...multiplied by independent rank one fast weights...
...yields ensemble weight matrices for each member.
$\mathbf{s}_1^\top$
$\circ$
$\mathbf{r}_1$
$\mathbf{r}_1\mathbf{s}_1^\top$
$=$
$W_1 = W \circ \mathbf{r_1 s_1}^T$
-1
0
1
W
$\mathbf{s}_2^\top$
$\circ$
$\mathbf{r}_2$
$\mathbf{r}_2\mathbf{s}_2^\top$
$=$
$W_2 = W \circ \mathbf{r_2 s_2}^T$
Figure 2: An illustration on how to generate the ensemble weights for two ensemble members.

Vectorization: We show how to make the above ensemble weight generation mechanism parallelizable within a device, i.e., where one computes a forward pass with respect to multiple ensemble members in parallel. This is achieved by the fact that manipulating the matrix computations for a mini-batch. Let x denote the activations of the incoming neurons in a neural network layer. The next layer's activations are given by:

$$\begin{aligned} y_n &= \phi\left(\overline{W}_i^\top x_n\right) & (2)\\ &= \phi\left(\left(W \circ r_i s_i^\top\right)^\top x_n\right) & (3)\\ &= \phi\left(\left(W^\top (x_n \circ s_i)\right) \circ r_i\right), & (4)\end{aligned}$$

where ϕ denotes the activation function and the subscript n represents the index in the mini-batch. The output represents next layer's activations from the i^{th} ensemble member. To vectorize these computations, we define matrices R and S whose rows consist of the vectors r_i and s_i for all examples in the mini-batch. The above equation is vectorized as:

$$Y = \phi\left(((X \circ S)W) \circ R\right). \tag{5}$$

where X is the mini-batch input. By computing Eqn. 5, we can obtain the next layer's activations for each ensemble member in a mini-batch friendly way. This allows us to take the full advantage of GPU parallelism to implement ensemble efficiently. To match the input and the ensemble weight, we can divide the input mini-batch into M sub-batches and each sub-batch receives ensemble weight $\overline{W}_i, i = \{1, \ldots, M\}$.

Ensembling During Testing: In our experiments, we take the average of predictions of each ensemble member. Suppose the test batch size is B and there are M ensemble members. To achieve an efficient implementation, one repeats the input mini-batch M times, which leads to an effective batch size $B \cdot M$. This enables all ensemble members to compute the output of the same B input data points in a single forward pass. It eliminates the need to calculate the output of each ensemble member sequentially and therefore reduces the ensemble's computational cost.

2.2 Computational Cost

The only extra computation in BatchEnsemble over a single neural network is the Hadamard product, which is cheap compared to matrix multiplication. Thus, BatchEnsemble incurs almost no additional computational overhead (Figure 1).[2] One limitation of BatchEnsemble is that if we keep the mini-batch size the same as single model training, each ensemble member gets only a portion of input data. In practice, the above issue can be remedied by increasing the batch size so that each ensemble member receives the same amount of data as ordinary single model training. Since BatchEnsemble is parallelizable within a device, increasing the batch size incurs almost no computational overhead in both training and testing stages on the hardware that can fully utilize large batch size. Moreover, when increasing the batch size reaches its diminishing return regime, BatchEnsemble can still take advantage from even larger batch size by increasing the ensemble size.

The only memory overhead in BatchEnsemble is the set of vectors, $\{r_1, \ldots, r_m\}$ and $\{s_1, \ldots, s_m\}$, which are cheap to store compared to the weight matrices. By eliminating the need to store full weight

[2]In Figure 1, note the computational overhead of BatchEnsemble at the ensemble size 1 indicates the additional cost of Hadamard products.

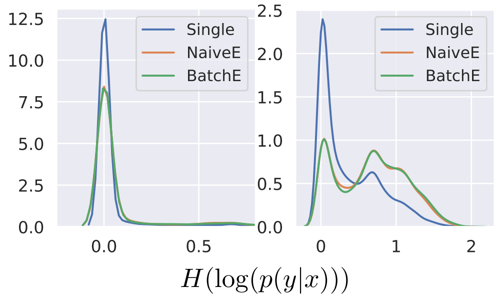
matrices of each ensemble member, BatchEnsemble has almost no additional memory cost. For example, BatchEnsemble of ResNet-32 of size 4 incurs 10% more parameters while naive ensemble incurs 4X more.

3 Experiments

3.1 Predictive Uncertainty

We first evaluate evaluates the predictive uncertainty of BatchEnsemble on out-of-distribution tasks and ECE loss. It is known that deep neural networks tend to make over-confident predictions even if the prediction is wrong or the input comes from unseen classes. Ensembles of models can give better uncertainty prediction when the test data is out of the distribution of training data. To measure the uncertainty on the prediction, we calculate the predictive entropy of single neural network, naive ensembles, and BatchEnsemble. The result is presented in Figure 3a. As we expected, single model produces over-confident predictions on unseen examples, whereas ensembling methods exhibit higher uncertainty on unseen classes, including both BatchEnsemble and naive ensemble. It suggests that BatchEnsemble doesn't inherit the desired uncertainty modelling of naive ensembles.

(a) Histogram of the predictive entropy on test examples from known classes, CIFAR-10 (left) and unknown classes, CIFAR-100 (right).



(b) Expected Calibration Error. Ensemble of size 4. Lower ECE reflects better calibration.

	MC-drop	BatchE	NaiveE	Single
C10	2.89	2.37	2.32	3.27
C100	8.99	8.89	6.82	9.28

Additionally, we calculate the Expected Calibration Error [Naeini et al., 2015] (ECE) of single model, naive ensemble and BatchEnsemble on both CIFAR-10 and CIFAR-100 in Table 3b. To calculate ECE, we group model predictions into M interval bins based on the predictive confidence (each bin has size $\frac{1}{M}$). Let B_m denote the set of samples whose predictive probability falls into the interval $(\frac{m-1}{M}, \frac{m}{M}]$ for $m \in \{1, \dots M\}$. Let $\text{acc}(B_m)$ and $\text{conf}(B_m)$ be the averaged accuracy and averaged confidence of the examples in the bin B_m. The ECE can de defined as the following,

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |\text{acc}(B_m) - \text{conf}(B_m)| \tag{6}$$

where n is the number of samples. ECE as a criteria of model calibration, measures the difference in expectation between confidence and accuracy [Guo et al., 2017]. It shows that BatchEnsemble makes more calibrated prediction compared to single neural networks.

We also evaluate the calibration of different mehtods on recently proposed CIFAR-10 corruption dataset [Hendrycks and Dietterich, 2019]. The dataset consists of over 30 types of corruptions to the images. It is commonly used to benchmark a wide range of methods on calibrated prediction [Ovadia et al., 2019]. To the best of our knowledge, dropout ensemble is the state-of-the-art memory efficient ensemble method. Thus, in our paper, we compare BatchEnsemble to dropout ensemble in this section. Naive ensemble is also plotted as an upper bound of our method. As showed in Figure 4, BatchEnsemble achieves better calibration than dropout as the skew intensity increases. Moreover, dropout ensemble requires multiple forward passes to get the best performance. Ovadia et al. [2019] used sample size 128 while we found no significant difference between sample size 128 and 8. Note that even the sample size is 8, it is 8X more expensive than BatchEnsemble in the testing time cost. Finally, we showed that combining BatchEnsemble and dropout ensemble leads to better calibration. It is competitive to naive ensemble while keeping memory consumption efficient. It is also an evidence that BatchEnsemble is an orthogonal method to dropout ensemble.

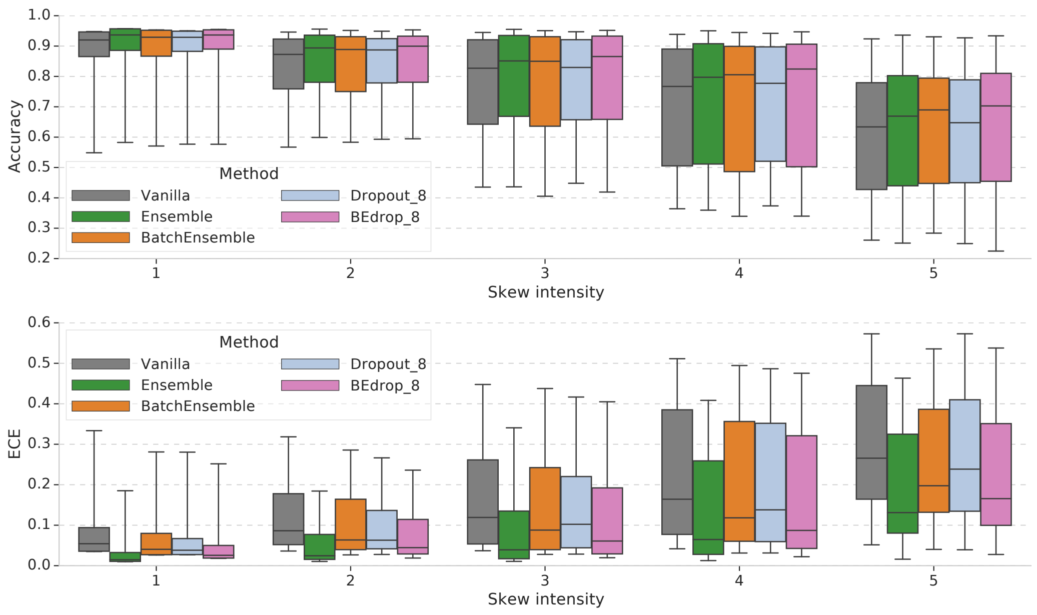


Figure 4: Calibration on CIFAR-10 corruptions: boxplots showing a comparison of ECE under all types of corruptions on CIFAR-10. Each box shows the quartiles summarizing the results across all types of skew while the error bars indicate the min and max across different skew types. **Ensemble/BatchEnsemble:** Naive/Batch ensemble of 4 ResNet32x4 models. **Dropout8:** Dropout ensemble with sample size 8. **BEdrop8:** BatchEnsemble of 4 models + Dropout ensemble with sample size 8.

3.2 Contextual Bandits

We evaluate our BatchEnsemble method on the recently proposed bandits benchmark [Riquelme et al., 2018]. Bandit data comes from different empirical problems that highlight several aspects of decision making. No single algorithm can outperform every other algorithm on every bandit problem. Thus, average performance of the algorithm over different dataset is used to evaluate the quality of uncertainty estimation. Thompson sampling [Thompson, 1933] is used in bandits problem to keep a model posterior distribution and select a greedy action according to the posterior of the model. Therefore, the key factor to achieve good performance in bandits problem with Thompson sampling is to learn a reliable uncertainty model. The uncertainty of ensembles can be represented by the variance of prediction over ensemble members. In our experiment, Thomson sampling samples from the policy given by one of the ensemble member. The fact that Dropout which is an implicit ensemble method achieves competitive performance on bandits problem suggests that ensemble can be used as uncertainty modelling. As the result in Table 1 showed, Both BatchEnsemble with ensemble size 4 and 8 outperform Dropout in terms of average performance.

4 Conclusion

We introduce BatchEnsemble, an efficient method for ensembling deep neural networks in a mini-batch friendly way. We show that BatchEnsemble has significantly less memory and computational cost (including training and testing) than naive ensemble. Moreover, we showed that BatchEnsemble produces comparable uncertainty estimation to naive ensemble—demonstrating that the slow and fast weight parameterization is sufficient for expressivity.

Table 1: Contextual bandits regret. Results are relative to the cumulative regret of the Uniform algorithm. We report the mean and standard error of the mean over 30 trials. Ensemble size with 4, 8.

	M.RANK	M.VALUE	MUSHROOM	STATLOG	FINANCIAL	JESTER	WHEEL
NaiveEnsemble4	**5.30**	34.64	**13.44 ± 3.83**	**7.10 ± 1.15**	11.31 ± 1.48	72.73 ± 6.32	68.63 ± 21.97
NaiveEnsemble8	6.50	34.91	13.59 ± 3.13	7.15 ± 0.98	11.64 ± 1.57	73.54 ± 6.14	68.63 ± 19.32
BatchEnsemble4	6.30	34.52	15.22 ± 5.21	11.53 ± 5.06	10.24 ± 2.66	72.65 ± 6.27	62.94 ± 26.12
BatchEnsemble8	5.70	**33.95**	13.48 ± 3.36	9.85 ± 3.67	13.17 ± 2.87	**71.84 ± 6.47**	61.41 ± 26.18
BBAlphaDiv	14.80	80.01	58.14 ± 4.13	69.78 ± 6.33	85.59 ± 4.61	89.04 ± 4.36	97.51 ± 10.95
BBB	12.20	44.35	23.48 ± 5.11	23.25 ± 5.18	33.54 ± 8.36	76.51 ± 6.27	64.99 ± 28.53
Dropout	8.20	36.73	15.05 ± 8.23	9.31 ± 3.19	13.53 ± 2.98	71.90 ± 6.31	73.86 ± 22.48
LinFullPost	9.40	49.60	97.42 ± 4.52	19.00 ± 1.03	10.24 ± 0.92	78.40 ± 4.85	**42.94 ± 12.68**
MultitaskGP	5.90	34.59	12.87 ± 4.70	8.04 ± 3.77	**8.50 ± 0.80**	74.03 ± 5.96	69.52 ± 18.55
NeuralLinear	10.40	35.61	15.49 ± 4.77	13.51 ± 1.30	17.58 ± 1.37	82.91 ± 3.55	48.56 ± 11.84
ParamNoise	10.40	36.84	16.45 ± 6.45	13.13 ± 3.37	14.89 ± 2.72	75.24 ± 6.57	64.47 ± 9.85
RMS	9.40	39.18	16.31 ± 6.13	10.44 ± 5.02	11.75 ± 2.64	73.38 ± 4.70	84.02 ± 24.67
Uniform Sampling	16.00	100.00	100.00	100.00	100.00	100.00	100.00

References

Léon Bottou. Stochastic learning. In *Summer School on Machine Learning*, pages 146–168. Springer, 2003.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2015.

Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pages 797–842, 2015.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

Lars Kai Hansen and Péter Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12:993–1001, 1990.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJz6tiCqYm.

Geoffrey E. Hinton and Radford M. Neal. Bayesian learning for neural networks. 1995.

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

Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in neural information processing systems*, pages 586–594, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*, pages 231–238, 1995.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS*, 2017.

Richard Maclin and David W. Opitz. Popular ensemble methods: An empirical study. *J. Artif. Intell. Res.*, 11:169–198, 1999.

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, 2015:2901–2907, 2015.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *ArXiv*, abs/1906.02530, 2019.

Michael P. Perrone and Leon N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. 1992.

Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown. 2018.

W.R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285—294, 1933.

Jingjing Xie, Bing Xu, and Chuang Zhang. Horizontal and vertical ensemble with deep representation for classification. *CoRR*, abs/1306.2759, 2013.