Proceedings of Machine Learning Research 95:176-191, 2018 ACML 2018

Hypernetwork-based Implicit Posterior Estimation and Model Averaging of Convolutional Neural Networks

Kenya Ukai UKAI@AI.CS.KOBE-U.AC.JP
Takashi Matsubara MATSUBARA@PHOENIX.KOBE-U.AC.JP
Kuniaki Uehara UEHARA@KOBE-U.AC.JP
Graduate School of System Informatics, Kobe University, 1-1 Rokko-dai, Nada, Kobe, Hyogo 657-8501, Japan

Editors: Jun Zhu and Ichiro Takeuchi

Abstract

Deep neural networks have a rich ability to learn complex representations and achieved remarkable results in various tasks. However, they are prone to overfitting due to the limited number of training samples; regularizing the learning process of neural networks is critical. In this paper, we propose a novel regularization method, which estimates parameters of a large convolutional neural network as implicit probabilistic distributions generated by a hypernetwork. Also, we can perform model averaging to improve the network performance. Experimental results demonstrate our regularization method outperformed the commonly-used maximum a posterior (MAP) estimation.

Keywords: hypernetwork, Bayesian estimation, convolutional neural network, image recognition

1. Introductions

Deep neural networks have a rich ability to learn complex representations and achieved remarkable results in various tasks; they surpassed humans in the performance on visual recognition (He et al., 2015) and speech recognition (Xiong et al., 2017, 2016). However, they are prone to overfitting due to the limited number of training samples; regularization of neural networks is an essential problem and has been investigated so far (see Bishop (2007), Section 5.5 and Goodfellow et al. (2016), Section 7 for a survey).

Many studies have addressed overfitting via parameter regularization (Goodfellow et al., 2016; Bishop, 2007). Weight decay prevents the weights from growing excessively large. It works as gradient descent on a quadratic weight term of the objective function and hence it is referred to as L2 regularization. Srivastava et al. (2014) proposed dropout to prevent units from excessively co-adapting by randomly dropping units from a neural network during training. Although these are heuristic methods, we can interpret them as Bayesian methods; L2 regularization is equivalent to the maximum a posteriori (MAP) estimation of the parameters with Gaussian priors (Bishop, 2007, Section 1.2.5), and dropout can be interpreted as the variational inference to minimize the Kullback-Leibler divergence to a Gaussian prior (Gal and Ghahramani, 2016). Also, many studies have trained the parameters of neural networks explicitly using Bayesian methods (Blundell et al., 2015; Krueger et al., 2017; Pawlowski et al., 2017); these models are called Bayesian neural networks

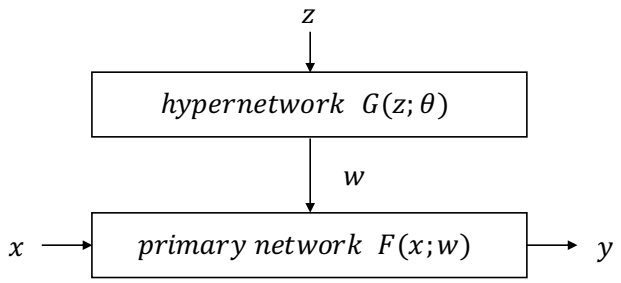


Figure 1: Structure of neural network using hypernetwork.

(BNNs) and have been used for a fully-connected network or small convolutional neural network (CNN). The BNNs restrict the variational posterior of parameters to a known distribution family and use drawn samples from the posterior for evaluation. Krueger et al. (2017) proposed Bayesian Hypernetworks to tackle the restriction. A hypernetwork is a neural network that outputs the parameters of another neural network (Ha et al., 2017). Thanks to the special architecture of the proposed hypernetwork, they calculated the exact divergence and modeled complicated shapes of the posterior distributions, but their performances are limited by Weight Normalization.

In this paper, we propose a new hypernetwork-based regularization method for large-scale CNNs. Like Krueger et al. (2017), we use hypernetworks to approximate the posterior of the parameters. Because of the difficulty applying to large networks, we do not employ generally-used variational inference; instead, we minimize the target likelihood directly. By inputting random samples to the hypernetworks, the outputs (i.e., the parameters of the target CNN) form a distribution implicitly (Goodfellow et al., 2014; Tran et al., 2017). This probabilistic behavior of the parameters regularizes the learning process.

We applied the proposed method to various CNNs. Experimental results demonstrate that our regularization method outperforms the MAP estimation under an appropriate setting. To our best knowledge, this is the first time for hypernetworks to build a deep CNN superior to the deep CNNs trained with the MAP estimation. Moreover, we perform Bayesian model averaging; this enables us to adjust the trade-off between the computational time and the performance.

2. Related Works

2.1. Hypernetworks

A hypernetwork is a neural network that outputs the parameters of another neural network (we call it "primary network", hereafter). Let $F(x; w) : X \times W \to Y$ be a primary network, where x is the input such as an image, y is the output such as a class label, and w is the parameters. A hypernetwork is defined as $G(z; \theta) : Z \times \Theta \to W$, where z is the input and θ is the parameters of the hypernetwork. Figure 1 shows the diagram. After training the primary network with hypernetworks, one obtains the optimal parameters w^* as the outputs of the hypernetworks instead of estimating w^* by minimizing the loss over the parameters w.

Using a hypernetwork, Ha et al. (2017) attempted to reduce the number of the parameters of wide residual networks (WideResNet) (Zagoruyko and Komodakis, 2016). They grouped every N parameters of the primary network and replaced each group with the output of the hypernetwork in response to a corresponding M-dimensional trainable input. When the number of hypernetwork's parameter is negligible, the total number of parameters were reduced N/M times. They employed the hypernetwork as a relaxed weight-sharing. However, the experimental results showed that the classification accuracy also got worse significantly. The results suggest that sharing one hypernetwork across the whole CNN is an excessively strong constraint.

2.2. Bayesian Neural Networks

When training a non-bayesian neural network, we define a loss function $\mathcal{L}(x, y, w)$, update the parameters w to minimize the loss function $\mathcal{L}$ by the gradient descent, and obtain the optimal parameters w^*. The cross-entropy $-\log p(y|x; w)$ is often used as the loss function. This scheme is the maximum likelihood estimation (MLE) (Goodfellow et al., 2016, Section 5.5). We can set prior distributions of the parameters, use $-\log p(w|x, y) = -\log p(y|x; w) - \log p(w) + C$ as the loss function, and obtain the optimal parameters w^*, where C is a constant value. This scheme is the maximum a posteriori (MAP) estimation (Goodfellow et al., 2016, Section 5.6). The MLE and MAP estimation are point estimations of the parameters.

Conversely, a Bayesian neural network is a neural network whose parameters are estimated as posterior distributions. One can regularize the learning process by the probabilistic behavior of the parameters and express uncertainty. In this case, one needs to integrate over the posterior of the parameters to calculate the exact output. Instead, the following Monte Carlo sampling is widely used;

$$p(y|x, \mathcal{D}) = \sum_{w \in \mathcal{W}} p(y|x, w, \mathcal{D}) p(w|\mathcal{D})$$

If the above Monte Carlo sampling is used in the evaluation, this is called Bayesian model averaging, creating an implicit ensemble of multiple models and improving the performance (Bishop, 2007, Section 14.1).

Blundell et al. (2015) proposed Bayes by Backprop, which enables backpropagation of neural networks over the Monte Carlo sampling of weight parameters. They used factorial Gaussian distributions as a prior and a variational posterior of each weight parameter. This strategy only captures a single mode of the true posterior, limiting the expression ability; Bayes by Backprop is inapplicable to deep CNNs directly (Krueger et al., 2017).

Krueger et al. (2017) proposed Bayesian Hypernetworks by employing hypernetworks to represent complex posteriors. They used invertible-structured neural networks called *real-NVP* (Dinh et al., 2017) as hypernetworks, which enables one to calculate the exact likelihood of a given parameter set. Real-NVP requires many layers and the same number of the input element as the output element; a simple application needs huge hypernetworks and numerous parameters. To prevent this issue, Krueger et al. (2017) employed Weight Normalization (Salimans and Kingma, 2016) and used hypernetworks only for generating scale parameters of Weight Normalization. Generally speaking, for image classification, the

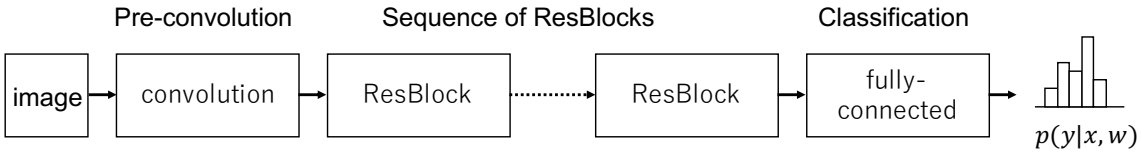


Figure 2: Structure of a ResNet (He et al., 2016)

performance of Weight Normalization is inferior to Batch Normalization, which is generally used for deep neural networks (Ioffe and Szegedy, 2015; Salimans and Kingma, 2016; Gitman and Ginsburg, 2017). As a result, the performances of Bayesian Hypernetworks are limited for image classification.

2.3. Convolutional Neural Networks

Convolutional neural networks (CNNs) perform well on image classification tasks. Especially, a residual network (ResNet) and its variants achieved state-of-the-art results (He et al., 2016; Han et al., 2017). Figure 2 shows the structure of a typical ResNet. A ResNet is composed of three parts: a pre-convolution layer, a sequence of residual blocks (ResBlocks), and a fully-connected layer. A ResBlock is composed of convolution layers, batch normalization layers, and activation functions; their order has some variations (He et al., 2016). Zagoruyko and Komodakis (2016) introduced WideResNet and improved the results by increasing the number of the channels. Xie et al. (2017) introduced ResNeXt and improved the performance by arranging ResBlocks in parallel. ResNet, WideResNet, and ResNeXt are suited for applying hypernetworks because their structures are simple; they use many ResBlocks of the same size repeatedly.

3. Proposed Regularization

3.1. Implicit Posterior Estimation of Parameters by Hypernetworks

We propose a regularization method for deep CNNs using hypernetworks. Unlike previous studies (Ha et al., 2017; Blundell et al., 2015; Krueger et al., 2017), we aim at a regularization of a comparative or superior performance to the MAP estimation.

As mentioned in previous sections, for a neural network, we minimize the loss function $\mathcal{L}(x, y, w)$, where x is an image, y is a true class label, and w is the parameters for image classification tasks. For a network with a hypernetwork $g(z;\theta)$, we train the hypernetwork's parameters θ instead of the primary networks' parameters w. We draw a sample z from a prior distribution $p(z)$ and input it to the hypernetwork. We used the output $g(z;\theta)$ as the primary network's parameters $w = g(z;\theta)$; the output w forms a distribution $q(w;\theta)$ implicitly (Goodfellow et al., 2014; Tran et al., 2017).

Ordinarily, the prior $p(w)$ and variational posterior $q(w)$ of the parameters w has been expressed as known distributions such as factorial Gaussian distributions (Blundell et al., 2015; Kingma and Welling, 2014). This assumption could constrain parameters excessively. Conversely, we do not assume the variational posterior as an explicit formulation of a density function, and instead, we use the implicit distribution $q(w;\theta)$ as the variational posterior. In other words, we consider the hypernetwork's output $w = g(z;\theta)$ given a

random sample $z \sim p(z)$ as a sample w from the variational posterior $q(w)$ of the primary network's parameters w. Thanks to this assumption, the primary network's parameters $q(w)$ could have a variational posterior of a complicated shape.

While the hypernetwork regularizes the weight parameters of the primary network, the hypernetwork should also be regularized. If not, the hypernetwork outputs arbitrary weight parameters and never restrict the behavior of the primary network. From a viewpoint of Bayesian model, this regularization corresponds to minimization of the divergence of the variational posterior from the prior. Following the common methodology, we introduce Gaussian priors to the hypernetworks' parameters and implemented the priors as the weight decay.

The joint distribution of the primary network and the hypernetwork is

$$p(y, x, \theta) = \int p(y|x, w)p(w|\theta)p(\theta)p(x)dw$$

where $p(w|\theta) = \int \delta(w = g(z; \theta))p(z)dz$. Here, we obtain the weight parameters w of the primary network as a sample drawn from the hypernetwork $p(w|\theta)$ like deep implicit generative models (Tran et al., 2017). Since $p(w|\theta)$ is not explicitly available, we approximate it by $\hat{p}(w|\theta) = \sum_{n=1}^{N} \delta(w = g(z_n; \theta))$, where $z_{1:N} \sim p(z)$. The approximate joint distribution is

$$\hat{p}(y, x, \theta) = \sum_{n=1, z_n \sim p(z)}^{N} p(y|x, w = g(z_n; \theta)))p(\theta)p(x)$$

We introduce a Gaussian prior $p(\theta)$ to the hypernetwork's parameter θ and get its approximate MAP estimation via weight decay (which is equivalent to L_2-regularization). Given an input x and the hypernetwork's parameter θ, the posterior probability of the class label y is

$$\hat{p}(y|x, \theta) = \sum_{n=1, z_n \sim p(z)}^{N} p(y|x, w = g(z_n; \theta))$$

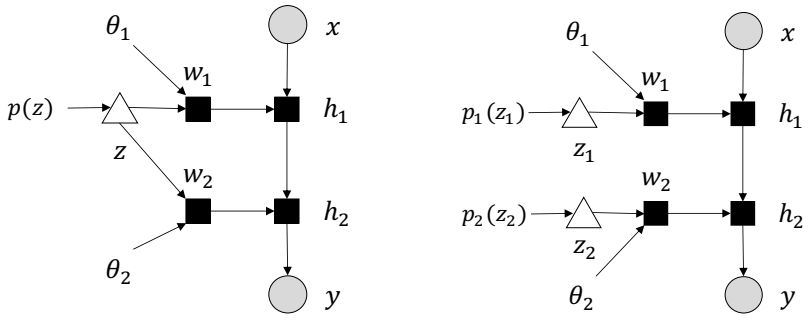
In our implementation, we use $N = 1$. Then, the loss function $\mathcal{L}$ is the cross-entropy of the true class label $p(y|x, \mathcal{D})$ obtained from the dataset $\mathcal{D}$ and the posterior $\hat{p}(y|x, \theta)$ in addition to the aforementioned L_2-regularization. The derivative of the loss function $\mathcal{L}$ w.r.t. the hypernetwork's parameters θ can be obtained by the backpropagation algorithm;

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial w} \frac{\partial w}{\partial \theta}\Big|_{w=g(z;\theta)}.$$

3.2. Parameter Generation

The hypernetworks build the higher-order relationships between weight parameters w. We propose two strategies for generating the parameters w.

First, we propose the *all-in-one* strategy, which builds the relationship among the whole parameters at once. Figure 3(*a*) shows the graphical model, where a deterministic variable is denoted by a black square, a sample from a probabilistic distribution is denoted by a white triangle, and an observed variable is denoted by a gray circle. In this strategy, the whole



(a) all-in-one strategy (b) block-wise strategy

Figure 3: Strategies for generating parameters.

parameters of the primary network are generated from a single sample z of the distribution $p(z)$; the whole parameters are related to each other.

Otherwise, following the previous studies (Ha et al., 2017; Krueger et al., 2017), we propose the *block-wise* strategy, dividing the parameters into N blocks and builds the relationship only inside each block. Figure 3(*b*) shows the graphical model for the case of $N = 2$. In this strategy, the parameters of each block i are generated from a sample z^i of distribution $p(z^i)$ independently from other blocks. Note that one can reduce the number of the parameters by sharing the hypernetwork among the blocks, but this results in an inferior performance (Ha et al., 2017).

For ResNets (He et al., 2016; Zagoruyko and Komodakis, 2016; Xie et al., 2017), we consider that each ResBlock is suited as the block. We employed a network of the same structure for the strategies. We prepared a hypernetwork $g(z; \theta_i)$ for each convolution layer i in the ResBlock of the primary network. In the *all-in-one* strategy, we drew a sample z from a distribution $p(z)$ and input it to all the hypernetworks. Then, each hypernetwork was co-adapted to each other. In the the *block-wise* strategy, we drew a sample from $p(z_i)$ for every ResBlock independently; this prevented ResBlocks from co-adapting.

When a convolution layer i in a ResBlock has a kernel of $f_{size} \times f_{size}$, an input of N_{in} channels, and an output of N_{out} channels, we built a hypernetwork $g_i(z_i; \theta_i)$ which outputs an $f_{size} \times f_{size} \times N_{in} \times N_{out}$-dimensional weight parameter. Note that the ResNets have no bias terms because of the batch normalization. We did not use hypernetworks for the components other than ResBlocks; the pre-convolution layer, the last fully-connected classification layer, and the batch normalization. Instead, we applied weight decay to them.

3.3. Model Averaging

Since we estimate the variational posterior of the parameters, we can perform model averaging in the evaluation. If samples from the learned distribution have a diversity, the model averaging improves the performance like an ensemble.

In this paper, we use different ways of the model averaging depending on the strategies. For the *all-in-one* strategy, we generate N sets of the parameters and build N primary networks. We perform the model averaging by averaging the outputs of the N networks. Figure 4(*a*) shows the diagram of a $\times 2$ model averaging for the all-in-one strategy. z^i denotes a sample from the prior $p(z)$ and transformed to parameters w_i by a hypernetwork.

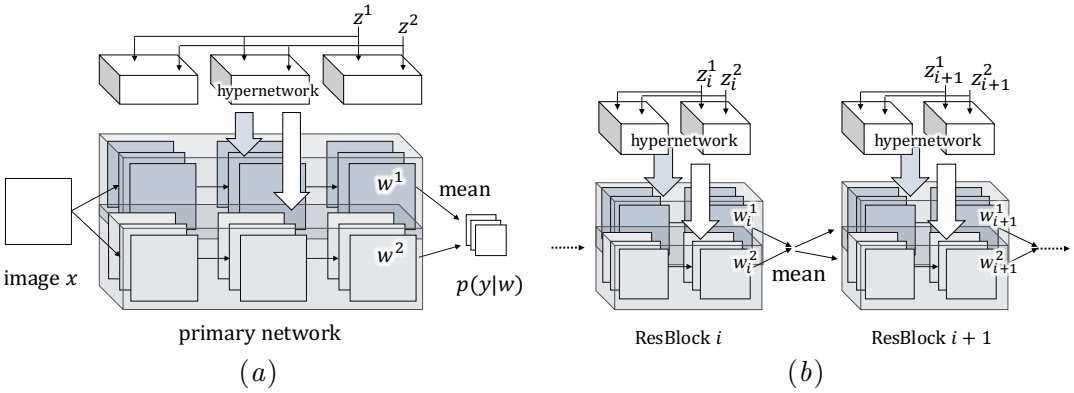


Figure 4: Model averaging of the (a) *all-in-one* and (b) *block-wise* strategies.

For the *block-wise* strategy, we generate N ResBlocks and arrange them in parallel like ResNeXt (Xie et al., 2017). Figure 4(b) shows the diagram of a $\times 2$ model averaging for the block-wise strategy. Note that this is not a strict form of model averaging mentioned in Section 2.2 because it averages the intermediate outputs instead of the final outputs.

3.4. Architecture of Hypernetwork

Each hypernetwork generates the weight parameters of the corresponding convolutional layer since each convolutional layer is a computational unit in CNNs. This setting surpass the total size of the hypernetworks and makes it available on our GPU server. When each hypernetwork generates the weight parameters of a resblock or the whole CNN, the total size of hypernetworks becomes much larger because of the explosion of combinations of weight parameters. We also examined the dimension numbers of the prior $p(z)$ and found that the results were robust to the dimension number as long as it was set to a number larger than a certain value (e.g., 8 in our experiments).

The increase in the computational complexity is almost negligible compared to that of the primary CNN at least under our experimental settings.

In the training phase, we drew a single weight set per mini-batch from a hypernetwork, and we used the weight set for all images in the mini-batch; this is the same strategy as Krueger et al. (2017). With an increase in the number of images per mini-batch, the computational complexity of the hypernetwork becomes negligible. We calculated the computational complexity following Canziani et al. (2016); a WideResNet 28-10 needs 5.24G Operations per 32×32 RGB image, and our proposed hypernetwork with $z_{dim} = 16$ increases the computational complexity by 0.59G Operations per mini-batch. Since the mini-batch size was 128, the increase in the computational complexity was just 0.1 %. Moreover, since we used a hypernetwork for each convolution layer, we can run the hypernetworks in parallel. However, the computation of the CNN has been highly optimized by the NVIDIA cuDNN library compared to that of the fully-connected networks used in the hypernetworks; a WideResNet 28-10 with our proposed hypernetwork needs around a computational time twice as long as the plain WideResNet. Once the library is optimized for the hypernetworks, we consider the time complexity becomes close to the theoretical computational complexity.

In the inference phase, if we do not perform model averaging, we can fix the drawn weight set and remove the hypernetworks. Then, the computational and time complexity is exactly the same as the plain CNN, and even in this case, our proposed methods performed better than plain CNNs in the experiments of *all-in-one* hypernetwork with the prior $p(z)$ of $\mathcal{U}(0, 1)$ for WideResNet28-10. The model averaging increases in the computational complexity proportionally to the number of involved models. We can adjust the trade-off between the computational complexity and the performance by determining the number of models.

4. Experiments and Results

4.1. Common Settings

We applied our regularization method to various CNNs based on residual networks (He et al., 2016). As a hypernetwork, we built a fully-connected feedforward neural network for each convolutional layer.

Each CNN layer has one feedforward hypernetwork, which has $z_{dim} * h_{dim} + h_{dim} * f_{size} * f_{size} * N_{in} * N_{out}$ parameters. We used $z_{dim} = h_{dim} = 32$ or $z_{dim} = h_{dim} = 16$ according to the limitation of computational resources.

We removed bias terms from all connections to remain the randomness of the hypernetwork inputs. We followed the training methods in the original study of each CNN unless otherwise stated.

Following the original studies (Zagoruyko and Komodakis, 2016; Han et al., 2017; Xie et al., 2017), all CNN architectures in our experiments were trained using batch normalization and weight decay and not using dropout unless otherwise stated. The weight parameters trained with the weight decay are equivalent to those with the L2 regularization and to the MAP estimates under the weight prior of a normal distribution (Bishop, 2007, Section 1.2.5). In other words, we refer to the CNNs trained with the weight decay as the MAP estimation.

For comparison, we trained the CNNs without the weight decay for evaluating the efficiency of the weight decay and our proposed method. Since the weight prior was removed, the resultant weight parameters can be regarded as the maximum likelihood estimation (MLE). Even in this case, we applied batch normalization. Note that Atanov et al. (2018) suggests that the batch normalization is related to the variational inference. If so, this case is not purely the MLE.

We initialized the parameters of hypernetworks using He's method (He et al., 2015) except for the final layers of hypernetworks. If the final layers initialized in the same way, we found that the loss function diverges to infinity at early stage of training owing to the large magnitude of weights drawn from hypernetworks and to the resultant large gradients. Instead, we initialized the final layers of hypernetworks with samples from uniform distribution whose bounds are 10 times smaller than He's method (He et al., 2015).

We evaluated the deterministic networks (i.e., the MLE and MAP estimation) by the median of three runs, which is a commonly used measure for deep CNNs (He et al., 2016). In addition, since our regularization method is stochastic, we evaluated it by the median of 15 runs (three models from scratch with weight generations from five different inputs).

4.2. Proposed Regularization Method and Model Averaging

We compared our regularization method of the *all-in-one* and *block-wise* strategies with the MLE and the MAP estimation as explained in Sec. 4.1 for WideResNet (Zagoruyko and Komodakis, 2016).

We followed the training methods in the original study of Zagoruyko and Komodakis (2016) unless otherwise stated. We evaluated the regularized WideResNet on CIFAR-10 dataset (Krizhevsky and Hinton, 2009); it consists of 50,000 training samples and 10,000 testing samples, and each sample is a 32×32 color natural image in 10 classes. We applied mean/std normalization, random horizontal flipping, padding, and random cropping to samples in the dataset. We trained all the parameters to minimize the cross-entropy loss by the stochastic gradient descendent (SGD) with a Nesterov momentum of 0.9 and a mini-batch size of 128. We set the learning rate to 0.1 and dropped by 0.2 at 40%, 60%, and 80% of the training period. The original study (Zagoruyko and Komodakis, 2016) trained the WideResNets for 200 epochs, but we found that a longer training further improves the test accuracies; we trained the WideResNets for 800 epochs. We applied the weight decay of 0.0005 to all the deterministic parameters, i.e., the parameters of hypernetworks and the parameters of the primary network not generated by hypernetworks. The hypernetworks' unit size was 32 for WideResNet28-4 and 16 for WideResNet28-10.

We employed the standard normal distribution $\mathcal{N}(0, 1)$ and the uniform distribution $\mathcal{U}(0, 1)$ as the prior $p(z)$ of the hypernetwork's input. Table 1 summarizes the test error rates, where $\times N$ implies averaging of N models. The numbers following the WideResNets are the hyperparameters, each of which determines the depth and width of the networks, respectively (see Zagoruyko and Komodakis (2016)). The MLE got the worse error rates; this emphasizes the importance of regularization. Without model averaging, the WideResNet generated by the hypernetworks with the all-in-one strategy and the prior $p(z)$ of $\mathcal{U}(0, 1)$ achieved the test error rates lower than that of the ordinary MAP estimation. This implies that our method with an appropriate prior successfully regularized the WideResNet and, to our best knowledge, this is the first time that a regularization by a hypernetwork outperformed the MAP estimation.

With model averaging, our regularization method achieved the lower test error rates in many cases. Especially, with the prior of $\mathcal{N}(0, 1)$ and the *block-wise* strategy, the model averaging improved the error rates by large margins. On the other hand, with the prior of $\mathcal{U}(0, 1)$, the improvement was limited. Figure 5 summarizes the test error rates with the varying number of averaged models.

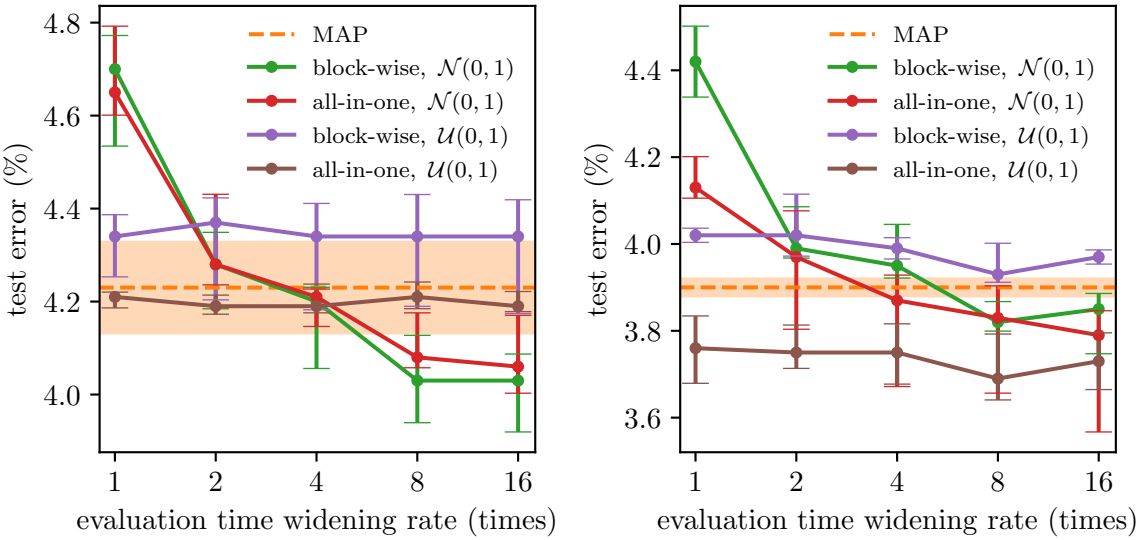
By definition, the model averaging employs the average of the outputs of multiple models, while the intermediate outputs were averaged in the *block-wise* strategy; this strategy is not guaranteed to function. Veit et al. (2016) showed that the ResNet behaves like an ensemble of relatively shallow structures (ResBlocks). In this sense, the *block-wise* strategy prevented co-adaptation between different blocks and was expected to enhance the ensemble within the ResNet. However, the *block-wise* strategy got worse test error rates than the *all-in-one* strategy, implying that a slight co-adaptation is required ever for ResNets. We will explore an appropriate co-adaptation in future works.

As another study related to our method, Ha et al. (2017) aimed at reducing the total number of parameters and, as they confess, the performance was severely worsened. Con-

Table 1: Test error rates on CIFAR-10 for WideResNet with different regularization methods.

			WideResNet28-4		WideResNet28-10	
Methods	strategy	prior $p(z)$	$\times 1$	$\times 16$	$\times 1$	$\times 16$
MLE	—	—	6.05%	—	5.49%	—
MAP	—	—	4.23%	—	3.90%	—
Baysian Hypernetworks	—	—	failed	—	failed	—
Bayes-by-Backprop	—	—	failed	—	failed	—
hypernetwork	all-in-one	$\mathcal{N}(0,1)$	4.65%	**4.06%**	4.13%	**3.79%**
	all-in-one	$\mathcal{U}(0,1)$	**4.21%**	**4.19%**	**3.76%**	**3.73%**
hypernetwork	block-wise	$\mathcal{N}(0,1)$	4.70%	**4.03%**	4.42%	**3.85%**
	block-wise	$\mathcal{U}(0,1)$	4.34%	4.34%	4.02%	3.97%

The error rates lower than that of the MAP estimation are emphasized using bold fonts.



(a) WideResNet28-4 (b) WideResNet28-10

Figure 5: The test error rates with the varying number of averaged models.

versely, our purpose is to regularize and improve the primary network; we consider that the comparison with Ha et al. (2017) is not fair.

We also examined weight normalization and confirmed that it does not work well for recent large-scale CNNs. The original study of weight normalization by Salimans and Kingma (2016) demonstrated that CNNs with weight normalization is inferior to CNNs with batch normalization for image recognition, while it works well for the fully-connected neural networks. Gitman and Ginsburg (2017) also confirmed a similar tendency. We applied weight normalization to WideResNet instead of batch normalization, and we confirmed that the weight parameters diverse to the infinity in the early phase of the training procedure.

Table 2: Double faults on CIFAR-10 of the WideResNet28-10 with different regularizations.

			number of missclassified samples			
Regularization method	strategy	prior $p(z)$	4 times	3 times	2 times	1 time
MAP estimation	—	—	172	114	129	258
hypernetwork	all-in-one	$\mathcal{N}(0,1)$	246	106	119	193
	all-in-one	$\mathcal{U}(0,1)$	340	15	29	32
	block-wise	$\mathcal{N}(0,1)$	216	119	123	223
	block-wise	$\mathcal{U}(0,1)$	327	50	43	66

We consider that weight normalization does not work well with recent large-scale CNNs or, at least, the appropriate hyper-parameters for the weight normalization are considerably different from those for bach normalization.

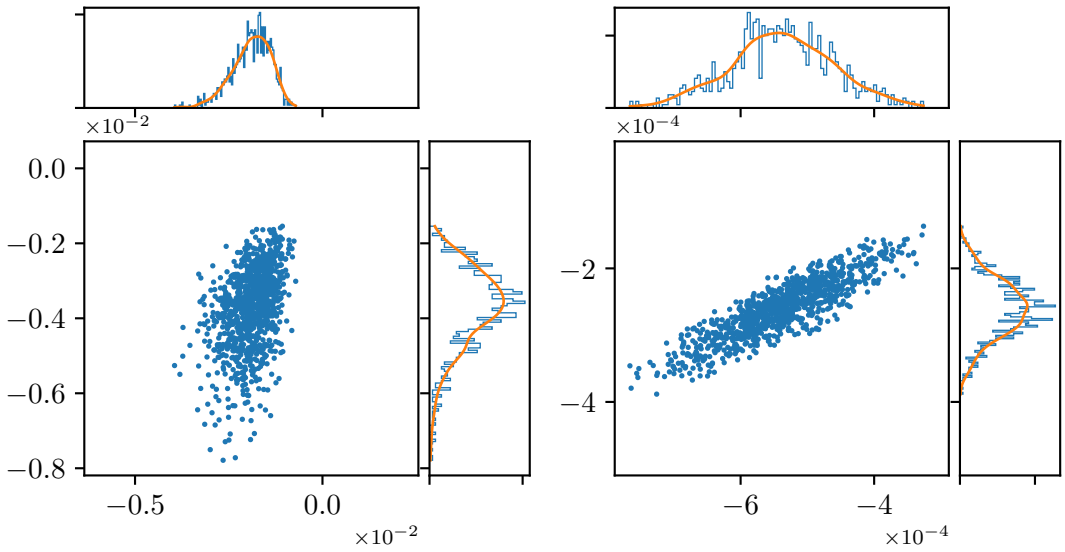
We also examined Bayesian hypernetwork (Krueger et al., 2017), but we found weight normalization disturbed the training procedure of WideResNet as stated above.

We also examined Bayes-by-Backprop and found that weight parameters continued fluctuating and the classification accuracy never exceeded the chance level. This was because the variance of drawn weight was much larger than their gradients. We consider that they are not suited for recent large-scale CNNs or we have to adjust hyper-parameters carefully unlike our proposed method. A more detailed adjustment is out of scope of this study.

4.3. Model Diversity

A model averaging is a Bayesian version of ensemble and, generally speaking, the ensemble improves the performances better if involved models have a greater diversity (Zhou, 2012). These results suggest that the prior $\mathcal{N}(0,1)$ generated a wide variety of the primary networks; some models got worse results, but the variety prompted the better performance after the model averaging. On the other hand, the prior $\mathcal{U}(0,1)$ achieved a better result without model averaging and got a limited improvement with model averaging, suggesting a limited variety of the primary networks.

We evaluated the diversity by double fault, which is defined as the ratio of test samples that two models misclassified together (Giacinto and Roli, 2001). A large ratio suggests that the two models classify samples with similar boundaries and have a limited diversity. We used four models (so it may be better to call it quadruple fault). Tables 2 summarizes the results of the double fault of the WideResNets. With our regularization method, we generated the WideResNets by the same hypernetwork with different inputs. For the MAP estimation, we trained the WideResNets from scratch. With the prior of $\mathcal{N}(0,1)$, the WideResNets got smaller ratios than the case with the prior of $\mathcal{U}(0,1)$, indicating a greater diversity. This is consistent with the performance improvement rates in Table 1. The ratios are at a similar level to the MAP estimation. Hence, using the prior of $\mathcal{N}(0,1)$, one can get multiple models from the hypernetworks and adjust the trade-off between the computational time and the performance by determining the number of models.



(*a*) *all-in-one*, $\mathcal{N}(0,1)$ (*b*) *all-in-one*, $\mathcal{U}(0,1)$

Figure 6: Scatter plot and histogram of samples from a approximate posterior of WideResNet28-10.

4.4. Weight Distributions

We examined the posterior distribution of weights learned by our proposed hypernetworks. Figure 6 shows the 1000 scatter samples from specific two dimensions of the weight posterior of WideResNet28-10 with the prior $\mathcal{U}(0,1)$ and $\mathcal{N}(0,1)$. We can find that the pair of weight parameters forms distributions with covariances (see Fig. 6). Conversely, the weight parameters learned with Bayes-by-Backprop have no covariance by definition and the MAP estimation is a point estimate.

We also calculated the correlation between random two dimensions and examined the difference of *all-in-one* strategy and *block-wise* strategy with the prior $\mathcal{U}(0,1)$. When we chose a pair of weight parameters randomly from the same ResBlock, the average of the absolute correlation was 0.977 for *all-in-one* strategy and 0.986 for *block-wise* strategy. When we chose a pair of weight parameters randomly from different ResBlocks, the average was 0.968 for *all-in-one* strategy and 0.008 for *block-wise* strategy. These results indicate that our proposed method trains stochastic weight parameters with correlations, and we can restrict the correlations across different subparts by choosing the strategies. According to the results in Table 1, the error rates were nonetheless insensitive to the strategies. With the prior of $\mathcal{N}(0,1)$, the correlations were relatively smaller (around 0.2 on average), but we can find higher order correlations and skewness as shown in Fig. 6(*a*). We conclude that our proposed method successfully obtains variational posterior that has a complicated shape compared to conventional methods.

Table 3: Test error rates on CIFAR-10 for WideResNet28-4 with different prior $p(z)$.

prior $p(z)$	$\times 1$	$\times 16$
$\mathcal{U}(0,1)$	4.21%	4.19%
$\mathcal{U}(-1,1)$	4.74%	4.22%
$\mathcal{N}(0,1)$	4.65%	4.06%
$\lvert\mathcal{N}(0,1)\rvert$	4.64%	4.58%

Table 4: Test error rates on CIFAR-10 for various CNNs.

	Pyramidal ResNet-110-48		ResNeXt-29-8-64d	
Methods	$\times 1$	$\times 16$	$\times 1$	$\times 16$
MAP	4.59%	—	4.03%	—
hypernetwork	4.64%	4.61%	**3.92%**	**3.91%**

4.5. Experiments with various Prior $p(z)$

We examine a normal distribution and a uniform distribution as the prior $p(z)$ of the hypernetworks. In preliminary experiments, we examined several intervals of the prior $p(x)$ and confirmed that the scale of the interval does not have a clear influence on the behavior of the primary network. This is because a neural network can scale the inputs by scaling the weight parameters of the input layer. Hence, we used typical values such as 0 and 1.

Also in preliminary experiments, we found the bias terms in the hypernetworks diminish the randomness of the hypernetwork inputs; we removed the bias terms. Because of not using bias terms in the hypernetworks, the hypernetworks cannot shift the activation arbitrarily. Hence, we examine the sign of the support by using the distributions $\mathcal{U}(0,1)$, $\mathcal{U}(-1,1)$, $\mathcal{N}(0,1)$, and $\mathcal{U}(0,1)$.

We performed the experiments on CIFAR-10 for WideResNet28-4 with *all-in-one* strategy. We summarized the results in Table 3. When employing the prior $p(z)$ of $\mathcal{U}(-1,1)$ and $\mathcal{N}(0,1)$, the accuracy was improved by model averaging; these distributions can take positive and negative values. Conversely, improvement by model averaging was limited with the prior distributions $\mathcal{U}(0,1)$ and $|\mathcal{N}(0,1)|$, which take positive values only. Hence, model averaging is effective when the prior takes both the positive and negative values, while the CNNs work better when the prior takes positive values only.

4.6. Experiments for Various CNNs

We also applied our proposed hypernetwork with the prior of $\mathcal{U}(0,1)$ to ResNeXt and Pyramidal ResNet on CIFAR-10. We summarized the results on Table 4.

As ResNeXt, we used *ResNeXt29-8-64d*, consisting of 29 convolution layers with cardinality of 8 and base channel widths of 64. As Pyramidal ResNet, we used *Pyramidal ResNet 110-48*, consisting of 110 convolution layers with widening factor of $\alpha = 48$. We employed *all-in-one* hypernetwork with the prior $p(z)$ of $\mathcal{U}(0,1)$, which worked better with WideRes-

Table 5: Test error rates on SVHN for WideResNet16-4.

Methods	prior $p(z)$	$\times 1$	$\times 16$
MAP	—	1.90%	—
hypernetwork	$\mathcal{U}(0,1)$	1.93%	1.93%
	$\mathcal{N}(0,1)$	1.90%	**1.80%**

Table 6: Test error rates on ImageNet for ResNet50 with the prior of $U(0,1)$.

	Top 1		Top 5	
Methods	$\times 1$	$\times 16$	$\times 1$	$\times 16$
MAP	23.84%	—	7.06%	—
hypernetwork	25.97%	25.87%	8.17%	8.16%

Net. The hypernetworks' unit size z was 16 for ResNeXt and 32 for Pyramidal ResNet. The hyper-parameters were the same as in the original studies except for the training epochs. We trained ResNeXt for 900 epoch and Pyramidal ResNet for 600 epoch.

We found that our proposed method achieved better results on ResNeXt and competitive results on Pyramidal ResNet. ResNeXt has a similar structure to WideResNet; it has multiple ResBlocks of the same channel size. Conversely, Pyramidal ResNet have ResBlocks of various channel sizes. Therefore, our proposed method could prefer a repetitive structure rather then Pyramidal ResNet.

4.7. Experiments for Various Datasets

We also evaluated our proposed method on SVHN (Russakovsky et al., 2015) using WideResNet16-4 (Zagoruyko and Komodakis, 2016). The hyper-parameters were the same as in Zagoruyko and Komodakis (2016). We employed *all-in-one* hypernetwork with the prior $p(z)$ of $\mathcal{U}(0,1)$ and $\mathcal{N}(0,1)$. The hypernetworks' unit size z was 32. We summarized the results in Table 5. Our proposed method with each prior was competitive to the baseline. With model averaging, our proposed method with the prior $p(z)$ of $\mathcal{N}(0,1)$ outperformed the baseline.

We also evaluated proposed method on ImageNet (Russakovsky et al., 2015). We used ResNet50 (He et al., 2016). The hyper-parameters were the same as in the original studies. We employed *all-in-one* hypernetwork with the prior $p(z)$ of $\mathcal{U}(0,1)$. The hypernetworks' unit size z is 32. We summarized the results in Table 6. The result is based on 1-run due to the limitation of computational resources.

The proposed method got slightly worse results than baseline. In the experiments on CIFAR-10 and SVHN datasets, the training error converged to almost zero while the test error had certain positive values; these results indicate slight overfitting of the CNNs. Conversely, in the experiments on ImageNet, the training error has remained at a similar level to the test error. This implies that the ResNet50 underfitted to the training set owing to its limited expression ability compared to the dataset size, and hence, it does not require

regularization. Huang et al. (2016) obtained a similar result. A larger network will be explored in future work.

5. Conclusions

We proposed a new regularization method for large-scale CNNs. We used hypernetworks to approximate posterior of the parameters under the implicit constraints. We applied the proposed regularization method to various CNNs. The probabilistic behavior of the parameters regularized the learning process. We evaluated four settings (strategies and priors $p(z)$) and found that our regularization method outperformed the MAP estimation under the appropriate settings. With model averaging, our regularization method improved their performance further. In future works, we will investigate the methodology to adjust the strength of co-adaptation and regularization.

Acknowledgments

This study partially supported by the JSPS KAKENHI (16K12487) and the MIC/SCOPE #172107101.

References

Andrei Atanov *et al.* Uncertainty estimation via stochastic batch normalization. In *ICLR Workshop*, 2018.

Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

Charles Blundell *et al.* Weight uncertainty in neural network. In *ICML*, 2015.

Alfredo Canziani *et al.* An analysis of deep neural network models for practical applications. *arXiv*, 2016.

Laurent Dinh *et al.* Density estimation using real NVP. In *ICLR*, 2017.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

Giorgio Giacinto and Fabio Roli. Design of effective neural network ensembles for image classification purposes. *Image Vision Comput.*, 2001.

Igor Gitman and Boris Ginsburg. Comparison of batch normalization and weight normalization algorithms for the large-scale image classification. *arXiv*, 2017.

Ian J. Goodfellow *et al.* Generative adversarial nets. In *NIPS*, 2014.

Ian J. Goodfellow *et al. Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

David Ha *et al.* Hypernetworks. In *ICLR*, 2017.

Dongyoon Han *et al.* Deep pyramidal residual networks. In *CVPR*, 2017.

Kaiming He *et al.* Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

Kaiming He *et al.* Identity mappings in deep residual networks. In *ECCV*, 2016.

Gao Huang *et al.* Deep networks with stochastic depth. In *ECCV*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, masters thesis, dept. of comp. sci., university of toronto. 2009.

David Krueger *et al.* Bayesian hypernetworks. *arXiv*, 2017.

Nick Pawlowski *et al.* Implicit weight uncertainty in neural networks. In *NIPS workshop on Bayesian deep learning*, 2017.

Olga Russakovsky *et al.* ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016.

Nitish Srivastava *et al.* Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

Dustin Tran *et al.* Hierarchical implicit models and likelihood-free variational inference. In *NIPS*, 2017.

Andreas Veit *et al.* Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, 2016.

Saining Xie *et al.* Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

Wayne Xiong *et al.* Achieving human parity in conversational speech recognition. *arXiv*, 2016.

Wayne Xiong *et al.* The microsoft 2016 conversational speech recognition system. In *ICASSP*, 2017.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.