Gradient Perturbation-based Efficient Deep Ensembles

Amit Chandak
amitch@cse.iitk.ac.in
Department of CSE, IIT Kanpur
India

Purushottam Kar
purushot@cse.iitk.ac.in
Department of CSE, IIT Kanpur
India

Piyush Rai
piyush@cse.iitk.ac.in
Department of CSE, IIT Kanpur
India

Abstract

Modern deep neural networks achieve excellent predictive performance due to their massive scale, flexible architecture design and availability of large training datasets. However, several applications additionally demand reliable estimates of model and predictive uncertainty that help in making robust predictions with limited training data, enabling out-of-distribution generalization, etc. Neural networks do not offer such uncertainly estimates *out-of-the-box*. Although Bayesian approaches to deep learning do provide a natural way to quantify model and predictive uncertainty by inferring the posterior distribution of the model weights and averaging the model's predictions over the entire posterior distribution, standard Bayesian inference methods such as MCMC and variational inference are difficult to design and scale to massive networks. An appealing and popular alternative is to learn an ensemble of model weights (popularly known as deep ensembles) and averaging the model's predictions over the ensemble. However, due to the need for multiple training runs, this approach also tends to be computationally expensive. In this work, we present PEG (Perturbed Ensemble via Gradient updates), a simple and efficient approach to constructing deep ensembles using gradients computed over validation data. Experiments show that PEG can not only create an ensemble provided a pre-trained model, but it can also further enrich pre-trained models that are deep ensembles themselves. On several benchmark datasets and architectures, PEG was found to perform favorably in comparison to state-of-the-art baselines in terms of predictive performance as well as other uncertainty quantification metrics.

CCS Concepts

• **Computing methodologies** → **Neural networks**.

Keywords

deep neural networks, ensemble learning, perturbation methods, approximate Bayesian inference

ACM Reference Format:
Amit Chandak, Purushottam Kar, and Piyush Rai. 2023. Gradient Perturbation-based Efficient Deep Ensembles. In *6th Joint International Conference on Data Science & Management of Data (10th ACM IKDD CODS and 28th COMAD) (CODS-COMAD 2023), January 4–7, 2023, Mumbai, India.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3570991.3571063

CODS-COMAD 2023, January 4–7, 2023, Mumbai, India

ACM ISBN 978-1-4503-9797-1/23/01…$15.00
https://doi.org/10.1145/3570991.3571063

1 Introduction

Deep neural networks (DNNs) have achieved breakthrough advances in a number of application domains including computer vision [20], natural language processing [35], speech processing [1], and robotics [27]. Recent advances have also shown how to train massive DNNs on large datasets achieving unprecedented performance on tasks such as conditional generation [31, 33]. Despite their strong predictive performance, these models usually lack the ability to systematically account for model uncertainty and predictive uncertainty. A key reason behind this is that these models are typically learned by optimizing a loss function on a large training set which yields a single solution (typically some form of approximate MAP or MLE) and ignores the uncertainty in the model weight estimates in the process. As a result, these models may yield predictions that are simultaneously over-confident and wrong [13]. Obtaining well-calibrated uncertainty estimates can be as important as having high predictive performance in specific domains such as autonomous vehicles [3] and healthcare [4] as over-confident, wrong predictions can have hazardous consequences in these applications.

A number of approaches have been proposed to estimate model and predictive uncertainty for DNNs. Taking a Bayesian approach [18] to learning DNN weights is a natural way to accomplish this wherein a posterior distribution is learnt over the DNN weights and predictions are made by averaging over the entire model posterior rather than relying on a single weight estimate. However, despite some promising efforts [9, 17], classic Bayesian inference methods such as MCMC [2] and variational inference [5, 38] are still considered difficult to design and scale for massive DNNs, hindering their widespread adoption. More recently, computationally cheaper approximations such as Monte-Carlo dropout [11], Laplace approximation [9, 32], or variants thereof [24, 26] have been explored. Monte-Carlo dropout takes a pre-trained model and creates an implicit ensemble out of it by randomly applying different dropouts at inference time, with each dropout pattern essentially giving a different version of the DNN model. On the other hand, Laplace approximation methods vary around a common idea of fitting a Gaussian centered around a local mode of the posterior, which usually corresponds to a stationary point of the loss function used to train the DNN. These techniques can perform approximate Bayesian inference and are more efficient. At inference time, averaging model predictions over the entire model posterior is intractable in general even if using Laplace approximations. Thus, it is common to further resort to Monte-Carlo averaging wherein a number of samples of model weights are sampled from the (Gaussian) posterior, predictions are computed using each sample, and then averaged. It is notable that Monte-Carlo methods of the kind discussed above are akin to using an ensemble [10] at inference time.

However, these approaches are often unable to capture multiple modes of the posterior since their ensembles are essentially perturbations around a single pre-trained model. This fact has motivated approaches that train an ensemble by exploring diverse initialization or hyper-parameter settings, and computing the ensemble-averaged predictions at inference time. Deep ensembles [21] is one such popular approach for constructing ensembles for DNNs by training the DNN with diverse initializations. An appealing aspect about ensembles constructed this way is that ensemble members can potentially capture multiple modes unlike methods such as Laplace approximation or Monte-Carlo dropout. However, deep ensembles are expensive to train since training must take place afresh for each member of the ensemble which is usually much more expensive than perturbation.

Our Contributions. In this work, we present PEG, an efficient approach to construct ensembles by applying *gradient-based* perturbations. PEG offers multiple benefits over existing methods:

(1) PEG is lightweight yet offers competitive or better performance than state-of-the-art methods both in terms of predictive performance as well as other uncertainty quantification metrics, at times outperforming expensive ensemble methods such as SWAG [24].
(2) Experiments suggest that PEG can not only create ensembles with better predictive performance out of a single pre-trained model, but it could enhance the performance of pre-trained ensembles such as a pre-trained deep ensemble [21] too.
(3) Being a gradient-based method naturally offers PEG benefits such as batch-normalization whereas Gaussian perturbation techniques have to carry out such steps separately at additional cost.
(4) Being gradient-based also allows PEG to explore variants such as those that perform unbiased updates. Experiments show that these variants can outperform the vanilla PEG method and are promising directions of future investigation.

2 Notation

Let $S = \{(x_i, y_i)\}_{i=1}^N$ denote the labeled training data – we will assume a multi-class classification task with C classes for sake of simplicity i.e. $y_i \in [C]$ for each data point $i \in [N]$. f will denote a neural architecture such as VGG-16, PreResNet-164, etc and θ will denote its parameters. We will use the shorthand $f_\theta(x)$ to denote the output of the neural architecture with model parameter θ on the data point x. For a multi-class tasks over C classes for which a penultimate softmax layer is used before the loss function, $f_\theta(x)$ will typically be a C-dimensional probability vector. $\mathbb{1}$ shall denote the indicator function so that $\mathbb{1}(A) = 1$ if event A happens or clause A is true else $\mathbb{1}(A) = 0$. The notation $p_\theta(y_i; x_i)$ will denote the label likelihood with model parameters θ. We will abuse notation to let $\log p_\theta(B)$ denote the average log-likelihood of the model parameter θ on the set of data points B

$$\log p_\theta(B) \triangleq \frac{1}{|B|} \sum_{i \in B} \log p_\theta(y_i; x_i)$$

Algorithm 1 PEG: Ensemble Creation and Inference

Require: Pre-trained base model weights θ, validation set V, ensemble size S, mini-batch size K, step-length η, Test point x_t
  $y_t \leftarrow \mathbf{0}$ //*Initialize the output vector*
  for $i = 1, \ldots, S$ **do**
    Draw a mini-batch B_i of K data points uniformly from V
    $\theta_i \leftarrow \theta - \eta \cdot \nabla_\theta p_\theta(B_i)$
    $y_t \leftarrow y_t + \frac{1}{S} \cdot f_{\theta_i}(x_t)$ //*Average ensemble predictions*
  end for
  return y_t

3 PEG: Perturbed Ensemble via Gradient Updates

In this section we outline the PEG method. Algorithm 1 presents the base PEG algorithm. PEG takes a pre-trained "base" model and applies gradient-based perturbations to create an ensemble. It is notable that if creating an ensemble of S models, PEG applies S perturbations but with *restart*. More specifically, the perturbations are not applied in succession but rather, all perturbations are applied to the same base model. This is a subtle yet crucial point – since the base model is expected to be a stationary point for the training loss function, applying perturbations successively rather than with restart was experimentally found to cause too much drift and poor performance.

Algorithm 1 considers a single test point. However, the procedure is readily modified to handle an entire batch of test points. PEG ensemble can even be archived to enable inference on a stream of test points. It is notable that PEG involves scarce hyper-parameters i.e., the step length η, tuned using a grid search.

PEG over Ensembles: PEG was found to be good at enriching ensemble models themselves. Algorithm 2 gives the pseudo code for this setting. Given an input ensemble with say R models, PEG can expand it to an ensemble of say $Q \cdot R$ models by perturbing each member of the ensemble. This technique was found to consistently give improvements in performance and indicates that techniques such as Deep Ensembles [21], although able to explore multiple modes efficiently, do not adequately explore the neighborhoods around those modes and stand to benefit from application of a second stage of ensemble creation. This two-stage ensemble creation can effectively be seen as achieving a multi-modal/multi-basin marginalization [37].

Unbiased Perturbations with PEG: Given its gradient-based perturbation method, on expectation, PEG makes the following perturbation to the base model (here V is the validation set)

$$\theta_i \leftarrow \theta - \eta \cdot \nabla_\theta p_\theta(V)$$

Given that the base model θ is a stationary point with respect to the training data, it is expected that $\nabla_\theta p_\theta(V) \to 0$. However, this may not be the case if the number of classes is large or if the validation set is not large enough. This is concerning since in these cases, PEG would induce a certain amount of drift to fit the validation set which may be undesirable. To investigate this further, we experimented with a variant of PEG that performs *provably* unbiased updates.

To do so requires us to obtain gradients on two mini-batches B_i^1, B_i^2 instead of one (per ensemble member) as well as obtain

Algorithm 2 PEG-Ensemble

Require: An ensemble model $\theta_1, \ldots, \theta_R$ with R pretrained models, validation set V, ensemble expansion factor Q, mini-batch size K, step-length η, Test point x_t

```
y_t ← 0                                   //Initialize the output vector
for j = 1, ..., R do                      //Apply PEG to each member of the input ensemble
    {θ_j^1, ..., θ_j^Q} ← PEG(θ_j, V, Q, K, η)
    for k = 1, ..., Q do
        y_t ← y_t + 1/(Q·R) · f_{θ_i^k}(x_t)
    end for
end for
return y_t
```

$\nabla_\theta p_\theta(V)$. Obtaining the latter may be a bit expensive but we can estimate $\nabla_\theta p_\theta(V)$ reasonably well using an average over several mini-batches. Given these, the unbiased version of PEG makes the following update

$$\theta_i \leftarrow \theta - \frac{\eta}{\sqrt{2}} \cdot (\nabla_\theta p_\theta(B_i^1) - \nabla_\theta p_\theta(B_i^2)) - a \cdot (\beta \cdot v + \eta \cdot \nabla_\theta p_\theta(V)),$$

where $a \sim N(0,1)$ is a Gaussian scalar variable, v is the momentum term, η is the step length as before and β is the momentum weight (typically set to 0.9). Straightforward calculations establish the following claim.

Claim 1. *For the unbiased update described above, we have*

(1) $\mathbb{E}[\theta_i - \theta] = 0$

(2) $\mathbb{E}[(\theta_i - \theta)(\theta_i - \theta)^\top] = \eta^2 \cdot D + \beta^2 \cdot vv^\top + \eta \cdot \beta \cdot (\nabla_\theta p_\theta(V) \cdot v^\top + v \cdot \nabla_\theta p_\theta(V)^\top)$

Proof (Sketch). The first part of the proof follows from linearity of expectation and the fact that $\mathbb{E}[\nabla_\theta p_\theta(B_i^1)] = \mathbb{E}[\nabla_\theta p_\theta(B_i^2)]$ and $\mathbb{E}[a|\theta, V, v] = 0$ since $a \sim N(0,1)$ independently. for the second part of the proof, the cross terms vanish similarly by using $\mathbb{E}[a|\theta, V, v] = 0$ and $\mathbb{E}[a^2|\theta, V, v] = 1$ as $a \sim N(0,1)$ independently. To simplify the homogeneous quadratic terms, we use $\mathbb{E}[\nabla_\theta p_\theta(B_i^1)\nabla_\theta p_\theta(B_i^1)^\top] = \mathbb{E}[\nabla_\theta p_\theta(B_i^2)\nabla_\theta p_\theta(B_i^2)^\top] \stackrel{\text{def}}{=} D$ and $\mathbb{E}[\nabla_\theta p_\theta(B_i^1)\nabla_\theta p_\theta(B_i^2)^\top] = \nabla_\theta p_\theta(V)\nabla_\theta p_\theta(V)^\top$ for $i = 1, 2$. □

Using a standard argument establishing SGD as a discrete-time approximation of a continuous-time Ornstein-Uhlenbeck (see [25] for example), PEG can also be seen as effecting a discretized OU-process to create its ensemble.

4 Related Work

A number of approaches have been proposed to improve the robustness of predictions of DNNs and to improve their uncertainty estimates. Broadly, these methods include (1) the classical Bayesian inference methods such as MCMC and variational inference which are usually intractable for DNNs but some recent works have explored their usage for DNNs as we discuss below; (2) Efficient Gaussian approximations of the DNN posterior which are either based on Laplace approximation [9, 32] or use SGD iterates to construct a Gaussian approximation of the posterior [24]; (3) deep ensembles [21] which are based on training multiple models, usually with different initialization, and have been shown to be equivalent to performing Bayesian model averaging; and (4) single forward-pass uncertainty estimation methods [28].

Bayesian inference for DNNs: Classical MCMC and variational inference methods are intractable for DNNs. Recent works have tried to adapt both MCMC and variational inference for DNNs. Among commonly used MCMC methods for DNNs, stochastic gradient based MCMC [39] have been used which are akin to SGD based optimization of DNNs with a Gaussian noise injected to each update. Recently, Hamiltonian Monte Carlo (HMC) based methods have also been applied for DNNs with some success [17]; however, they require significant computation budget. Among commonly used variational inference methods for DNNs, methods such as Bayes-by-backprop [6] and probabilistic backpropagation [14], methods that are based on introducing weight perturbation in natural gradient descent to yield a variational posterior [19]. However, both MCMC and variational inference methods are still considered difficult to design and tune for DNNs.

Gaussian approximations of the DNN posterior: Recent work on Bayesian inference for DNNs has also focused on obtaining cheaper approximations of the DNN weight posterior. One such commonly used approximation is a Gaussian centered around the maximum-a-posteriori (MAP) solution θ_{MAP} of the DNN weights. The classical Laplace approximation which approximates the posterior by a Gaussian $\mathcal{N}(\theta|\theta_{MAP}, \mathbf{A})$ where $\mathbf{A}$ is the inverse Hessian of the negative log-posterior, and more recently its efficient versions [9, 32] too, has been employed for DNNs with some success. Another recent approach along the same lines is SWAG [24] which approximates the DNN weight posterior by a Gaussian $\mathcal{N}(\theta|\theta_{SWA}, \mathbf{A})$, where θ_{SWA} is the solution obtained by running the Stochastic Weight Averaging (SWA) algorithm [16], and $\mathbf{A}$ is a covariance matrix (full/diagonal) constructed using SWA iterates.

Deep Ensembles and its variants: Deep ensembles [21] are based on training a DNN with multiple times with each run using a different initialization or different hyperparameters, and have been shown to be a promising way to improve the robustness of DNN predictions, and to obtain reliable uncertainty estimates. Although deep ensembles may appear non-Bayesian at first blush, they have been shown to perform Bayesian model averaging [37]. Another appealing aspect of deep ensembles is that they can capture the multiple basins of attraction [37] more easily than classical Bayesian inference methods applied to DNNs (and note that, simpler Bayesian inference methods such as Gaussian approximation of the posterior can only capture one basin of attraction). Deep ensembles however can be expensive since they require training the model multiple times. Monte-Carlo dropout [11] is a faster way to construct a deep ensemble from a pre-trained model by applying multiple random dropouts, with each dropout giving a different model. Other methods to generate deep ensembles faster include M-heads [22], Snapshot Ensembles [15], Fast Geometric Ensemble [12], and Batch Ensemble [36]. These approaches, usually considered *implicit ensembles*, do not need a separate training for each ensemble members but require a single run to sequentially generate all the ensemble members, or share parameters across the ensemble members with each member having a separate head. Another method PEP (Parameter Ensembling by Perturbation) provides a fast way to construct a deep ensemble by sampling multiple models from a Gaussian

centered at a solution θ_* obtained by training the DNN using a standard optimizer [26]. PEP is somewhat similar in spirit to our approach of using gradient perturbation to generate an ensemble. However, PEP uses a spherical Gaussian to generate the ensemble members and the spherical variance parameter needs to be tuned carefully to achieve the best performance.

Single forward-pass uncertainty estimation: Recent works such as DUQ [34] and SNGP [23] take a fundamentally different approach and try to estimate model/predictive uncertainty via a single forward pass [28] by using distance-aware output layers. However, these methods require changes to the training procedure as well as additional hyperparameters to be introduced.

Our approach PEG provides a simple and efficient way to construct ensembles from a single pre-trained model (e.g., SWA) or can also be used to enrich an already learned deep ensemble [21] by capturing local diversity around each mode captured by the original deep ensemble. Thus it complements methods such as SWA and deep ensembles by offering a simple, plug-in mechanism to further improve their performance without incurring much additional cost since PEG does not require any re-training.

5 Experiments

We report empirical evaluation of PEG by leveraging it as a wrapper around 3 methods, SWA [16], deep ensembles [21] and a version of SWAG [24], all of which also serve as baselines, in addition to PEP [26] which is another parameter perturbation based ensemble method like PEG. We take the model offered by each of these 3 methods as a pre-trained model and use PEG to generate an ensemble. In case of Deep ensemble, the solution is already an ensemble, using Algorithm 2 we make a fixed number of perturbations to each member and get an enriched and expanded ensemble.

We compare various methods on following metrics: Negative Log Likelihood (NLL), Classification Error (CE), Brier Score (BS) and Expected Calibration Error (ECE). NLL and CE are measures of the model's predictive performance whereas BS and ECE are measures of the model's calibration performance which is an indicative of how reliable the model's predictive uncertainties are. For all these metrics, a method with lower value is considered better. BS and ECE are briefly described below. Our code is available here.

Brier Score: [7] It measures the accuracy of predictive probabilities in classification tasks and is a proper scoring rule – recall that C is the number of classes in the multi-class task. The Brier score is computed as the average mean squared distance between predicted class probabilities and one-hot class labels:

$$\text{BS} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{C}\sum_{c=1}^{C}(p_\theta(\mathrm{y}_i = c|\mathrm{x}_i) - \mathbb{1}[\mathrm{y}_i = c])^2$$

Expected Calibration Error (ECE): [29] It measures the difference between predictive confidence and empirical accuracy in classification. ECE divides model's predictions into bins depending on their predictive probability. After dividing the [0,1] range into a set of bins $\{H_s\}_{s=1}^{S}$, it weights the miscalibration in each bin by the number of points that fall into it $|H_s|$:

$$\text{ECE} = \sum_{s=1}^{S}\frac{|H_s|}{N}\text{acc}(H_s) - \text{conf}(H_s))$$

where

$$\text{acc}(H_s) = \frac{1}{|H_s|}\sum_{\mathrm{x}_i \in H_s}\mathbb{1}[\mathrm{y}_i = \underset{c}{\text{argmax}}\, p_\theta(\mathrm{y}_i = c|\mathrm{x}_i)] \quad \text{and}$$

$$\text{conf}(H_s) = \frac{1}{|H_s|}\sum_{\mathrm{x}_i \in H_s}\max_c p_\theta(\mathrm{y}_i = c|\mathrm{x}_i).$$

Note that in the definition of BS and ECE, when using a Bayesian approach or ensemble, the quantity $p_\theta(\mathrm{y}_i|\mathrm{x}_i)$ is replaced by the posterior-averaged or ensembled-averaged probabilities.

Hyperparameters: In our experiments, we consider three network architectures, VGG-16, PreResNet-164 and WideResNet-28x10, on CIFAR-10 and CIFAR-100 datasets. We use PyTorch code and parameter settings provided with the SWAG [24] implementation for training these networks to get the baseline models like deep ensembles, SWA, SWAG-Diag and SWAG. We split the CIFAR training set randomly in a ratio of 80:20 to create the final train and validation set. We then train the various models on the final training set. We use the validation set to choose the perturbation parameter values of PEP and the perturbation learning rate for PEG, i.e., σ and η, respectively. PEG does a grid search among the values $(0.001, 0.01, 0.1, 1)$ for optimal η value. PEP [26] uses *golden section search* method [30] to find optimal σ in the range $(5e-4, 5e-3)$. We use their code to compute σ in our experiments.

In Sec. 5.4 we also apply PEG in a transfer learning setting where the models are trained on CIFAR-10 and tested on STL-10.

5.1 PEG with SWA pre-training

In our first experiment, we apply PEG to generate an ensemble using a pre-trained model obtained by SWA [16]. We compare the performance of this ensemble with SWA and PEP as baselines. For all the architectures considered, SWA is trained using SGD. The results are shown in Table 1 for CIFAR-10 and in Table 2 for CIFAR-100. In these tables, PEG-15 and PEP-15 indicate that an ensemble of 15 perturbed models is created from the SWA solution. To create an ensemble of PEG with 15 perturbed models, we first estimate the optimal η using grid search on the SWA solution with $S = 15$. This gives us 15 perturbed models whose predictions are then averaged and η with best metrics is chosen. For inference (test-time), Algorithm 1 is used with the optimal η and $S = 15$ on the SWA solution. $K = 128$ is chosen as the mini-batch size for all the experiments. PEP-15 ensembles are created similarly. For PEG-30 and PEP-30, $S = 30$ is chosen. As shows in Table 1 and Table 2, PEG performs better than SWA and is competitive with or better than PEP, both in terms of predictive performance as well as calibration metrics. Figure 1 shows ECE plots for SWA, ,PEP and PEG. Bin-width is 0.1 and each bin should have at-least 10 elements.

5.2 PEG with Deep Ensembles pre-training

In this experiment, we apply PEG to a pre-trained deep ensemble and perturb each of them to generate a richer ensemble. We compare the performance of this ensemble with the original pre-trained ensemble and PEP as the baselines. To generate the original pre-trained ensemble, for each network architecture, 3 models are trained with different random seeds using SGD. These 3 solutions form the baseline DeepEns-3. For inference (test-time), their predictions are averaged to compute required metrics. As before, PEG-15 and PEP-15 indicate that an ensemble of 15 perturbed models is

Table 1: CIFAR-10 - SWA as the base solution

	CIFAR-10				
Model	**PEG-15**	**PEP-15**	**PEG-30**	**PEP-30**	**SWA**
	NLL				
VGG-16	0.251	0.257	**0.244**	0.255	0.266
PreResNet-164	0.157	0.153	0.155	**0.151**	0.16
WideResNet28x10	0.111	0.112	**0.111**	0.112	0.113
	Classification Error %				
VGG-16	**6.78**	6.87	6.8	6.85	6.9
PreResNet-164	4.33	4.33	4.33	**4.32**	4.34
WideResNet28x10	3.74	3.76	**3.73**	3.78	3.81
	Brier				
VGG-16	0.107	0.108	**0.107**	0.107	0.11
PreResNet-164	0.068	0.067	0.068	**0.067**	0.069
WideResNet28x10	0.055	0.055	**0.055**	0.056	0.056
	ECE %				
VGG-16	**3.508**	3.966	3.573	3.878	4.275
PreResNet-164	2.248	2.149	2.198	**2.042**	2.347
WideResNet28x10	**0.885**	1.016	0.895	1.058	1.128

Table 2: CIFAR-100 - SWA as the base solution

	CIFAR-100				
Model	**PEG-15**	**PEP-15**	**PEG-30**	**PEP-30**	**SWA**
	NLL				
VGG-16	1.228	1.336	**1.163**	1.33	1.389
PreResNet-164	0.77	0.77	0.771	0.771	0.773
WideResNet28x10	0.691	0.689	0.69	**0.684**	0.715
	Classification Error %				
VGG-16	26.88	**26.81**	26.97	26.82	26.9
PreResNet-164	20.72	20.91	**20.72**	20.83	20.8
WideResNet28x10	**18.36**	18.37	18.43	18.41	18.43
	Brier				
VGG-16	0.398	0.416	**0.391**	0.415	0.424
PreResNet-164	0.3	0.301	**0.3**	0.301	0.301
WideResNet28x10	0.266	0.265	0.265	**0.264**	0.27
	ECE %				
VGG-16	12.616	15.363	**11.441**	15.26	16.373
PreResNet-164	**7.78**	8.014	7.838	8.043	8.07
WideResNet28x10	6.266	6.264	6.435	**6.134**	7.597

created from DeepEns-3. To create an ensemble of PEG with 15 perturbed models, as described in Sec. 5.1, we first estimate the optimal η using grid search on each member of DeepEns-3 with $S = 5$. This gives us 15 perturbed models whose predictions are then averaged and η with best metrics is chosen. For inference (test-time), Algorithm 1 is used with optimal η and $S = 5$ on each member of DeepEns-3. $K = 128$ is chosen as the mini-batch size for all the experiments. PEP-15 ensembles are created similarly. For PEG-30 and PEP-30, $S = 10$ is chosen for each member of DeepEns-3. As shown in Table 3 and Table 4, PEG performs better than DeepEns and is competitive or better than PEP, both in terms of predictive performance and calibration metrics. In Table 5 and Table 6, we also compare Deep Ensembles with PEG on a single base model whose solution was obtained by running SGD.

5.3 PEG with SWAG-Diag pre-training

We apply PEG to each member of an ensemble generated by sampling from the Gaussian (with diagonal covariance) posterior approximation of SWAG-Diag [24]. SWAG-Diag is less expressive

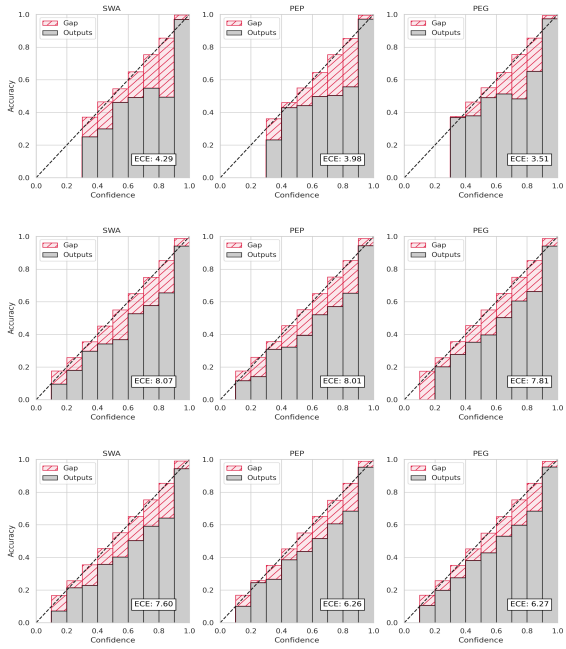


Figure 1: ECE plots of SWA, PEP & PEG for VGG16 on CIFAR-10 , PreResNet-164 & WideResNet28x10 on CIFAR100

Table 3: CIFAR-10 - DeepEns as the base solution

	CIFAR-10				
Model	**PEG-15**	**PEP-15**	**PEG-30**	**PEP-30**	**DeepEns-3**
	NLL				
VGG-16	0.225	0.229	**0.219**	0.221	0.232
PreResNet-164	0.139	0.135	0.135	**0.132**	0.14
WideResNet28x10	0.11	0.11	0.11	0.11	0.11
	Classification Error %				
VGG-16	6.31	**6.01**	6.3	6.07	6.09
PreResNet-164	**4.11**	4.21	4.11	4.14	4.13
WideResNet28x10	3.42	3.42	**3.41**	3.45	3.36
	Brier				
VGG-16	0.095	0.095	**0.094**	0.094	0.095
PreResNet-164	0.063	0.063	0.063	**0.062**	0.063
WideResNet28x10	0.051	0.051	0.051	0.051	0.051
	ECE %				
VGG-16	1.836	1.963	**1.469**	1.818	2.16
PreResNet-164	0.999	0.745	0.63	**0.581**	1.06
WideResNet28x10	0.684	0.604	0.704	**0.601**	0.636

approximation than SWAG which uses a full-rank covariance matrix. The purpose of this experiment is to show that perturbing each member of the SWAG-Diag with PEG improves the performance of generated ensemble and whether it performs comparably/better than SWAG.

SWAG-Diag model is trained as described in [24]. For inference (test-time prediction) with SWAG and SWAG-Diag, the model weights are sampled from Gaussian with full-rank and diagonal covariance matrix, respectively. Using the sampled weights, Monte-Carlo averaging is performed for predictions. In our experiments,

Table 4: CIFAR-100 - DeepEns as the base solution

	CIFAR-100				
Model	**PEG-15**	**PEP-15**	**PEG-30**	**PEP-30**	**DeepEns-3**
	NLL				
VGG-16	1.156	1.174	**1.059**	1.139	1.233
PreResNet-164	0.726	0.708	0.712	**0.703**	0.737
WideResNet28x10	0.714	0.716	**0.709**	0.714	0.712
	Classification Error %				
VGG-16	25.34	25.07	**24.81**	24.98	25.19
PreResNet-164	19.24	**19.13**	19.18	19.22	19.36
WideResNet28x10	17.71	17.92	**17.67**	17.9	17.86
	Brier				
	0.356	0.358	**0.347**	0.354	0.362
PreResNet-164	0.274	0.271	0.272	**0.270**	0.274
WideResNet28x10	0.259	0.259	**0.258**	0.259	0.259
	ECE %				
VGG-16	6.653	7.644	**2.1**	7.088	8.53
PreResNet-164	1.836	1.824	**1.548**	1.606	2.854
WideResNet28x10	5.047	4.77	5.005	**4.07**	4.466

Table 5: CIFAR-10: DeepEns Vs (PEG on SGD). Starting from a single SGD model (DeepEns-1), PEG outperforms it comfortably and given enough ensemble members, approaches the performance of DeepEns-3 too, indicating PEG's ability to offer some mode-exploration benefits.

	CIFAR-10				
Model	**PEG-10**	**PEG-30**	**PEG-50**	**DeepEns-1**	**DeepEns-3**
	NLL				
VGG-16	0.314	0.279	0.256	0.333	0.232
PreResNet-164	0.179	0.177	0.171	0.192	0.14
WideResNet28x10	0.135	0.133	0.133	0.136	0.11
	Classification Error %				
VGG-16	7.28	7.32	7.26	7.18	6.09
PreResNet-164	4.8	4.76	4.78	4.73	4.13
WideResNet28x10	3.78	3.81	3.79	3.77	3.36
	Brier				
VGG-16	0.118	0.113	0.112	0.12	0.095
PreResNet-164	0.075	0.075	0.075	0.077	0.063
WideResNet28x10	0.06	0.06	0.059	0.06	0.051
	ECE %				
VGG-16	4.569	2.072	2.476	4.907	2.16
PreResNet-164	2.117	1.923	1.642	2.55	1.06
WideResNet28x10	1.683	1.638	1.6	1.651	0.636

we generate 15 samples of the weights for computing SWAG and SWAG-Diag predictions and these serve as our baselines for this experiment. As described in Sec. 5.1 and 5.2, to create an ensemble of PEG with 15 perturbed models, we first estimate the optimal η using grid search on each sample of SWAG-Diag with $S = 1$ (i.e., a single perturbation). This gives us 15 perturbed models whose predictions are then averaged and η with best metrics is chosen. During inference (test-time), for each sample of model weights from SWAG-Diag, Algorithm 1 is used with optimal η and $S = 1$ to perturb them. $K = 128$ is chosen as mini-batch size for all the experiments. PEP-15 ensembles are created similarly. As shown in Table 7 and Table 8, applying PEG on SWAG-Diag generated ensemble indeed yields better results than the original SWAG-Diag based ensemble in most cases. PEG is also competitive or better than PEP and performs comparably or better than SWAG (which uses a full covariance) in most cases. Note that our main comparison here is not meant to be with SWAG but to show that PEG can improve the base solution given by SWAG-Diag. Also note that, PEG and PEP are lightweight as compared to SWAG since they only require the diagonal elements of covariance matrix.

Table 6: CIFAR-100: DeepEns Vs (PEG on SGD) - similar to Tab 5

	CIFAR-100				
Model	**PEG-10**	**PEG-30**	**PEG-50**	**DeepEns-1**	**DeepEns-3**
	NLL				
VGG-16	1.465	1.347	1.321	1.833	1.233
PreResNet-164	0.905	0.868	0.861	0.983	0.737
WideResNet28x10	0.797	0.793	0.792	0.833	0.712
	Classification Error %				
VGG-16	28.67	28.3	28.37	28.57	25.19
PreResNet-164	22.46	22.34	22.39	22.43	19.36
WideResNet28x10	19.69	19.66	19.56	20.27	17.86
	Brier				
VGG-16	0.415	0.405	0.409	0.469	0.362
PreResNet-164	0.322	0.315	0.315	0.335	0.063
WideResNet28x10	0.283	0.282	0.282	0.293	0.051
	ECE %				
VGG-16	7.1	8.387	10.765	20.165	8.53
PreResNet-164	7.725	5.828	6.272	10.691	2.854
WideResNet28x10	4.255	4.186	4.133	4.656	4.466

Table 7: CIFAR-10 - SWAG-Diag as base solution

	CIFAR-10			
Model	**PEG-15**	**PEP-15**	**SWAG-Diag**	**SWAG**
	NLL			
VGG-16	**0.224**	0.231	0.225	0.212
PreResNet-164	**0.134**	0.136	0.214	0.137
WideResNet28x10	0.113	**0.112**	0.146	0.119
	Classification Error %			
VGG-16	6.66	**6.63**	6.69	6.81
PreResNet-164	4.37	**4.25**	6.85	4.55
WideResNet28x10	3.84	**3.72**	4.66	3.9
	Brier			
VGG-16	0.101	0.102	0.101	0.1
PreResNet-164	**0.064**	0.065	0.101	0.066
WideResNet28x10	**0.056**	0.056	0.071	0.059
	ECE %			
VGG-16	2.523	2.782	2.501	1.156
PreResNet-164	**0.698**	1.016	0.765	0.667
WideResNet28x10	0.417	**0.403**	0.597	0.998

5.4 Transfer Learning

We also evaluate PEG for the task of transfer learning to investigate its efficacy against dataset shifts at test time. For this experiment, we consider the problem of transferring a model trained on CIFAR-10 for testing it on the STL-10 dataset [8, 24]. For this we use PEG based ensemble constructed via the three approaches considered earlier, i.e., (1) SWA as base solution, (2) deep ensembles as base solution, and (3) SWAG-Diag as base solution. Table 9, Table 10, and Table 11 show the results for these 3 cases. PEG performs better than PEP when SWA or DeepEns-3 is base solution.

Table 8: CIFAR-100 - SWAG-Diag as base solution

	CIFAR-100			
Model	**PEG-15**	**PEP-15**	**SWAG-Diag**	**SWAG**
	NLL			
VGG-16	1.094	**1.059**	1.076	1.037
PreResNet-164	0.723	**0.722**	0.953	0.731
WideResNet28x10	0.658	**0.656**	0.776	0.655
	Classification Error %			
VGG-16	26.58	26.62	26.58	26.5
PreResNet-164	21.2	**21.06**	26.75	21.07
WideResNet28x10	**18.3**	18.34	21.25	18.6
	Brier			
VGG-16	0.377	0.369	0.367	0.374
PreResNet-164	**0.292**	0.292	0.365	0.295
WideResNet28x10	0.259	**0.258**	0.301	0.261
	ECE %			
VGG-16	8.886	5.121	7.546	4.893
PreResNet-164	1.235	**0.979**	3.474	3.88
WideResNet28x10	3.677	3.257	2.612	1.844

Table 9: Transfer Learning CIFAR to STL: SWA as the base solution

	CIFAR-10				
Model	**PEG-15**	**PEP-15**	**PEG-30**	**PEP-30**	**SWA**
	NLL				
VGG-16	1.346	1.384	1.324	1.331	1.403
PreResNet-164	1.378	1.371	1.38	1.366	1.41
WideResNet28x10	1.012	1.01	1.002	1.012	1.022
	Classification Error %				
VGG-16	28.412	28.55	28.425	28.463	28.562
PreResNet-164	24.2	24.3	24.2	24.175	24.187
WideResNet28x10	22.962	22.888	22.85	22.862	22.912
	Brier				
VGG-16	0.459	0.472	0.46	0.462	0.476
PreResNet-164	0.399	0.397	0.398	0.397	0.403
WideResNet28x10	0.354	0.353	0.353	0.354	0.355
	ECE %				
VGG-16	19.171	20.504	19.401	19.518	20.881
PreResNet-164	16.697	16.628	16.67	16.484	17.135
WideResNet28x10	12.706	12.741	12.53	12.811	13.073

Table 10: Transfer Learning CIFAR-10 to STL : DeepEns as base solution

	CIFAR-10				
Model	**PEG-15**	**PEP-15**	**PEG-30**	**PEP-30**	**DeepEns-3**
	NLL				
VGG-16	1.316	1.324	1.267	1.287	1.343
PreResNet-164	1.199	1.154	1.142	1.136	1.226
WideResNet28x10	1.013	0.977	1.013	0.958	1.01
	Classification Error %				
VGG-16	27.375	27.438	27.25	27.412	27.463
PreResNet-164	24.1	24.062	24.187	24.087	24.125
WideResNet28x10	22.962	23.038	22.938	23.162	22.9
	Brier				
VGG-16	0.419	0.424	0.414	0.42	0.425
PreResNet-164	0.374	0.369	0.368	0.367	0.375
WideResNet28x10	0.357	0.351	0.356	0.349	0.355
	ECE %				
VGG-16	14.959	15.598	14.38	15.241	15.905
PreResNet-164	13.081	12.431	12.295	12.231	13.382
WideResNet28x10	12.434	11.753	12.389	11.47	12.276

Table 11: Transfer Learning CIFAR-10 to STL : SWAG-Diag as base solution

	CIFAR-10			
Model	**PEG-15**	**PEP-15**	**SWAG-Diag**	**SWAG**
	NLL			
VGG-16	1.245	1.226	1.196	1.104
PreResNet-164	1.21	1.169	1.108	1.047
WideResNet28x10	0.983	0.944	0.925	0.862
	Classification Error %			
VGG-16	28.638	28.425	28.588	28.038
PreResNet-164	24.425	24.487	26.013	25.113
WideResNet28x10	23.462	23.738	24.187	23.562
	Brier			
VGG-16	0.444	0.44	0.439	0.414
PreResNet-164	0.379	0.373	0.395	0.371
WideResNet28x10	0.351	0.349	0.359	0.342
	ECE %			
VGG-16	17.512	16.894	16.802	13.302
PreResNet-164	13.841	12.765	13.159	12.46
WideResNet28x10	11.542	10.545	11.818	9.793

5.5 Unbiased Update

Results of the unbiased update approach is shown in Table 12. This approach gives significant improvement over the vanilla PEG method but requires us to compute the gradient over the entire validation set (or an estimate thereof using multiple mini-batches).

Table 12: Unbiased update with SWA as the base solution

	CIFAR-10			CIFAR-100		
Model	**PEG-Unbiased**	**PEG-30**	**SWA**	**PEG-Unbiased**	**PEG-30**	**SWA**
	NLL			**NLL**		
VGG-16	**0.243**	0.244	0.266	**1.108**	1.163	1.389
PreResNet-164	**0.152**	0.155	0.16	**0.708**	0.771	0.763
WideResNet28x10	**0.111**	0.111	0.113	**0.682**	0.69	0.715

5.6 Additional Experiments on PEG vs PEP

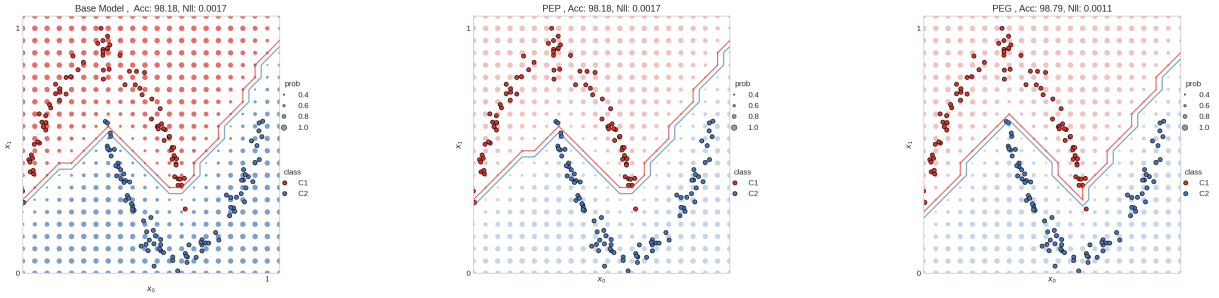
PEG and PEP enrich an existing model by generating an ensemble using perturbations. PEP generates ensemble of model-weights by sampling from a Gaussian with spherical covariance-matrix. This assumption is quiet restrictive and PEG overcomes it by using gradient information, thus ensuring model-weights belongs to the sub-space with low objective value and high accuracy. We demonstrate this by comparing PEP and PEG on two-dimensional synthetic datasets:

5.6.1 *Linearly Separable Dataset*

Assume model weights $W \in \mathbb{R}^2$ follow a Gaussian distribution. If data of two classes are distributed as in Fig. 4 then ensembles of W using PEP will have lower accuracy since PEP generates the model weights from a spherical Gaussian whereas PEG generates the model weights using gradient information.

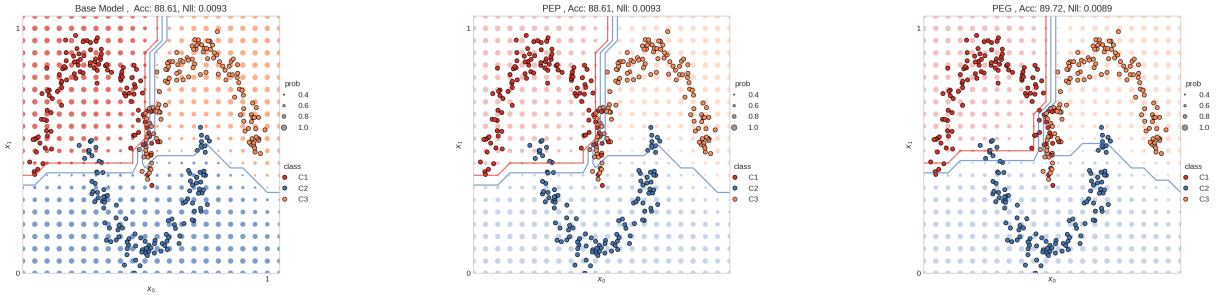
5.6.2 *Nonlinearly Separable Dataset: 3 Moons*

We generate 3-Moons data and train a Multi-Layer Perceptron (MLP) (1-hidden layer with 20 neurons) which yields an accuracy of 88.6% and a negative log-likelihood (NLL) of 0.0093. This MLP is now enriched



(a) Base Model, Acc: 98.18, NLL: 0.0017 **(b) PEP, Acc: 98.18, NLL: 0.0017** **(c) PEG, Acc: 98.79, NLL: 0.0011**

Figure 2: PEG offers visually better classifiers than PEP with higher accuracy and lower NLL



(a) Base Model, Acc: 88.6, NLL: 0.0093 **(b) PEP, Acc: 88.6, NLL: 0.0093** **(c) PEG, Acc: 89.72, NLL: 0.0089**

Figure 3: PEG offers visually superior classifiers than PEP with higher accuracy and lower NLL

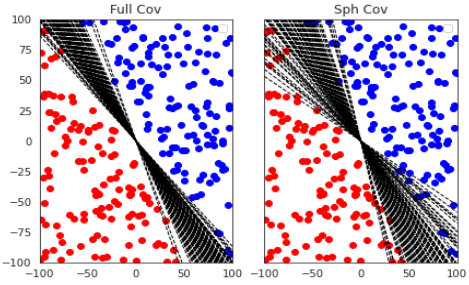


Figure 4: Models generated by PEG (left) vs PEP (right). PEP samples from spherical Gaussian, violates the margin and intrude into data territory whereas PEG models remain restricted within the margin.

by generating 5 ensemble members using PEG and PEP. Single perturbation in both the algorithm doesn't improve predictive performance but PEG with multiple perturbations improves the accuracy whereas doing the same for PEP does not improve the accuracy as ensemble members hardly move away from the base model due to a small optimal value of the variance (given by the golden search procedure) of the Gaussian. Results are shown in Fig. 3a to 3c

5.6.3 ***Nonlinearly Separable Dataset: 2 Moons*** We generate 2 Moons data and train a MLP (1-hidden layer with 20 neurons , accuracy: 98.18 and negative log-likelihood (NLL): 0.0017). This base-model is now enriched by generating 5 ensembles using PEG or PEP. We observe a similar behavior for PEG vs PEP as for the 3 Moons dataset. The results are shown in Fig. 2a to 2c

6 Conclusion

We presented PEG, a simple and lightweight approach to generate an ensemble by applying gradient-based perturbations around a given pre-trained model. PEG can also be used to enrich a pre-trained *ensemble* of models by applying a gradient-based perturbation around each member of the ensemble. In either case, PEG offers improvements in predictive and uncertainty metrics, over the base pre-trained model or model ensemble.

References

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*. PMLR, 173–182.

[2] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. 2003. An introduction to MCMC for machine learning. *Machine learning* 50, 1 (2003), 5–43.

[3] Fabio Arnez, Huascar Espinoza, Ansgar Radermacher, and François Terrier. 2020. A comparison of uncertainty estimation approaches in deep learning components for autonomous vehicle applications. *arXiv preprint arXiv:2006.15172* (2020).

[4] Edmon Begoli, Tanmoy Bhattacharya, and Dimitri Kusnezov. 2019. The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence* 1, 1 (2019), 20–23.

[5] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association* 112, 518 (2017), 859–877.

[6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. Weight uncertainty in neural network. In *International conference on machine learning*. PMLR, 1613–1622.

[7] Glenn W. Brier. 1950. VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY. *Monthly Weather Review* 78 (1950), 1–3.

[8] Adam Coates, Andrew Ng, and Honglak Lee. 2011. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 215–223.

[9] Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. 2021. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems* 34 (2021), 20089–20103.

[10] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*. Springer, 1–15.

[11] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *ArXiv* abs/1506.02142 (2016).

[12] T. Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. 2018. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. *ArXiv* abs/1802.10026 (2018).

[13] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International conference on machine learning*. PMLR, 1321–1330.

[14] José Miguel Hernández-Lobato and Ryan Adams. 2015. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International conference on machine learning*. PMLR, 1861–1869.

[15] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. Snapshot Ensembles: Train 1, get M for free. *ArXiv* abs/1704.00109 (2017).

[16] Pavel Izmailov, Dmitrii Podoprikhin, T. Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. 2018. Averaging Weights Leads to Wider Optima and Better Generalization. *ArXiv* abs/1803.05407 (2018).

[17] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. 2021. What are Bayesian neural network posteriors really like?. In *International conference on machine learning*. PMLR, 4629–4640.

[18] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. 2022. Hands-on Bayesian neural networks—A tutorial for deep learning users. *IEEE Computational Intelligence Magazine* 17, 2 (2022), 29–48.

[19] Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. 2018. Fast and Scalable Bayesian Deep Learning by Weight-Perturbation in Adam. In *ICML*.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[21] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf

[22] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David J. Crandall, and Dhruv Batra. 2015. Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks. *ArXiv* abs/1511.06314 (2015).

[23] Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan. 2020. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems* 33 (2020), 7498–7512.

[24] Wesley J. Maddox, T. Garipov, Pavel Izmailov, Dmitry P. Vetrov, and Andrew Gordon Wilson. 2019. A Simple Baseline for Bayesian Uncertainty in Deep Learning. In *NeurIPS*.

[25] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. 2017. Stochastic Gradient Descent as Approximate Bayesian Inference. *Journal of Machine Learning Research* 18 (2017), 1–35.

[26] Alireza Mehrtash, Purang Abolmaesumi, Polina Golland, Tina Kapur, Demian Wassermann, and William Wells. 2020. Pep: Parameter ensembling by perturbation. *Advances in Neural Information Processing Systems* 33 (2020), 8895–8906.

[27] Eduardo F Morales, Rafael Murrieta-Cid, Israel Becerra, and Marco A Esquivel-Basaldua. 2021. A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning. *Intelligent Service Robotics* 14, 5 (2021), 773–805.

[28] Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip HS Torr, and Yarin Gal. 2021. Deep Deterministic Uncertainty: A Simple Baseline. *arXiv e-prints* (2021), arXiv–2102.

[29] Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. 2015. Obtaining Well Calibrated Probabilities Using Bayesian Binning. *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence* 2015 (2015), 2901–2907.

[30] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007. Numerical Recipes 3rd Edition: The Art of Scientific Computing.

[31] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* (2022).

[32] Hippolyt Ritter, Aleksandar Botev, and David Barber. 2018. A Scalable Laplace Approximation for Neural Networks.. In *ICLR (Poster)*. OpenReview.net. http://dblp.uni-trier.de/db/conf/iclr/iclr2018.html#RitterBB18

[33] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. 2022. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *arXiv preprint arXiv:2205.11487* (2022).

[34] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. 2020. Uncertainty estimation using a single deep deterministic neural network. In *International conference on machine learning*. PMLR, 9690–9700.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[36] Yeming Wen, Dustin Tran, and Jimmy Ba. 2020. BatchEnsemble: An Alternative Approach to Efficient Ensemble and Lifelong Learning. *ArXiv* abs/2002.06715 (2020).

[37] Andrew G Wilson and Pavel Izmailov. 2020. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems* 33 (2020), 4697–4708.

[38] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. 2018. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence* 41, 8 (2018), 2008–2026.

[39] Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. 2019. Cyclical Stochastic Gradient MCMC for Bayesian Deep Learning. In *International Conference on Learning Representations*.