Task Agnostic Continual Learning Using Online Variational Bayes

Chen Zeno* **Itay Golan*** **Elad Hoffer** **Daniel Soudry**
Technion - Israel Institute of Technology, Haifa, Israel
{itaygolan, elad.hoffer, daniel.soudry}@gmail.com chenzeno@campus.technion.ac.il

Abstract

Catastrophic forgetting is the notorious vulnerability of neural networks to the change of the data distribution while learning. This phenomena has long been considered a major obstacle for allowing the use of learning agents in realistic continual learning settings. Although this vulnerability of neural networks is widely investigated, it is currently only mitigated by explicitly reacting to the change of task. We suggest a novel approach for overcoming catastrophic forgetting in neural networks, using an online version of the variational Bayes method. Having a confidence measure of the weights alleviates catastrophic forgetting and, for the first time, succeeds in this even without the knowledge of when the tasks are being switched.[2]

1 Introduction

Deep Neural Networks (DNNs) have been successfully used for solving a variety of tasks such as images classification [8], speech recognition [2], natural language processing [19] and more. However, those networks were trained and tuned on a single task. Often, a desirable property for a network is the ability to retain performance on previous tasks even when presented with new ones.

Catastrophic forgetting [22] is the tendency of neural networks to completely and rapidly forget previously learned information when learning new information, such as when changing a task or a source, as in continual learning. In such continual learning tasks we are interested in training a neural network over multiple tasks $D_1, ..., D_K$ sequentially. More formally, in continual learning the tasks arrive one after the other:

$$\begin{aligned} d_{1:T_1} &\in D_1 \\ d_{T_1+1:T_2} &\in D_2 \\ &\dots \\ d_{T_{K-1}+1:T_K} &\in D_K \end{aligned}$$

where d_i denote the input data feds into the network on iteration i. Thus, after iteration T_j the input data switches from D_J (Task J) to D_{J+1} (Task $J+1$). Note, that we assume the optimizer has access only to the current task's data. Evaluation is performed, over all seen tasks, once training on the final task has concluded. In this configuration, the standard approaches suffer from catastrophic forgetting; they perform well on latest tasks, and poorly on tasks from the early stages of training.

We split continual learning algorithms into two categories:

*Equal contribution

[2]Source code to replicate our results throughout the paper will be available online.

Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montréal, Canada.

1. **Task-aware:** The algorithm requires to be informed if the task is being switched in order to take some special action (e.g. adjust the parameters of the cost function after each task).
2. **Task-agnostic:** The algorithm is unaware of the tasks schedule, so it cannot take any special action, it has no knowledge that the data distribution changed.

Task-aware algorithms have a significant disadvantage, as in many realistic applications we may not possess the knowledge about the tasks schedule. Hence, task-agnostic continual learning is a harder setup as less information is provided to the system. Current algorithms rely heavily on this information, which enables them to take significant actions if needed.

Catastrophic forgetting was extensively investigated over the course of the past decades and various methods to overcome this problem were suggested. It was already observed 30 years ago that estimating the underlying posterior distribution can be beneficial to combat this phenomena [22]. Lately, it was found that estimating confidence of the weights through their posterior, and using it to affect their plasticity may allow a natural transition between learned tasks, while reducing the ill effect of catastrophic forgetting [14, 25]. However, existing methods for reducing catastrophic forgetting do require the knowledge of when the tasks are switched ("task-aware").

Contributions. In this work, we present Bayesian Gradient Descent (BGD) – a novel approach to overcome catastrophic forgetting in neural networks. We use online variational Bayes method to update the posterior every mini-batch with a closed-form update rule. A notable strength of our approach, compared to previous methods, is that our method is task-agnostic — it is able to adapt without an explicit indication that something has changed. BGD results are competitive with various task-aware continual learning algorithms, despite being task-agnostic and not taking into account interactions between the weights (a diagonal posterior method). Expanding BGD to non-diagonal (omitting the mean field approximation) is not trivial, but our results imply that it can boost BGD's performance.

2 Related work

Continual learning is a major challenge to existing artificial intelligence algorithms which are based on DNNs. Several approaches were suggested to allow continual learning. Some of them alter the network architecture, some add a regularization term and some use Bayesian inference. To the best of our knowledge, all of the current algorithms are task-aware.[3] [26] provides an extensive review of continual learning methods. We focus on approaches which do not use any samples from previous tasks.

One approach to prevent catastrophic forgetting is to design new architectures of neural networks. [28] suggested freezing and expanding the model on every task switch to prevent catastrophic forgetting, however, it is impractical for a large number of tasks due to the memory growth with each task. [29] proposed a dual model architecture consisting of a deep generative model that acts as a memory and a task solving model, a new generative model is created when the task is switched. Methods such as these, where the architecture is progressively modified, will not be at the focus of our work.

Another approach to prevent catastrophic forgetting is adding a regularization term to the loss function. "Elastic weight consolidation" (EWC) proposed by [14], slows the conversion of parameters important to the previous tasks by adding a quadratic penalty on the difference between the optimal parameter of the previous task and the current parameter. The importance of each weight is measured using the diagonal of the Fisher information matrix of the previous tasks. "Synaptic Intelligence" (SI) proposed by [35] also used a quadratic penalty on the difference between the optimal parameter of the previous task and the current parameter. However, the importance of each weight is measured as the path length of the updates on the previous task. [18] use knowledge distillation so the network with of the previously learned tasks is enforced to be similar to the network of the current task.

The Bayesian framework provides a solution to continual learning challenge in the form of Bayes' rule. When data arrives sequentially, the posterior distribution of the parameters for the previous task is used as a prior for the new task. "Variational Continual Learning" (VCL) proposed by [25] used online variational inference combined with the standard variational Bayes approach developed by

[3] A trivial task-agnostic algorithm is SGD/ADAM, which experiences severe catastrophic forgetting.

[4], "Bayes By Backprop" (BBB) to reduce catastrophic forgetting by replacing the prior on a task switch. In the BBB approach, a "mean-field approximation" is applied, meaning that a fully factorized distribution is used to approximate the posterior distribution. Thus, VCL is a diagonal method. [27] suggest using Bayesian online learning with a Kronecker factored Laplace approximation to attain a non-diagonal method for reducing catastrophic forgetting, which allows the algorithm to take into account interactions between weights within the same layer. Our focus in this work is on Bayesian diagonal methods.

We discuss additional related work on variational Bayes methods in Appendix C.

We summarize the major differences between our contributions and the previous works as follows:

1. **Task-agnostic vs. task-aware:** BGD (our method) is the first algorithm, to the best of our knowledge, to tackle catastrophic-forgetting in a task-agnostic scenario.
2. **Prior replacement:** BGD replaces the prior on each mini-batch during training, as opposed to methods such as VCL which replaces it at each task switch.
3. **Closed form update rule:** We update the posterior over the weights in closed form, in contrast to BBB (and VCL which relies on BBB) which use SGD optimizer on the variational free energy.
4. **Scalability:** Unlike Vanilla BBB, BGD is scalable and easy to implement not only on MNIST, but also on CIFAR10.

3 Theory

The process of Bayesian inference requires a full probability model providing a joint probability distribution over the data and the model parameters. The joint probability distribution can be written as a product of two distributions:

$$p\left(D,\boldsymbol{\theta}\right)=p\left(D|\boldsymbol{\theta}\right)p\left(\boldsymbol{\theta}\right)\ , \tag{1}$$

where $p\left(D|\boldsymbol{\theta}\right)$ is the likelihood function of the data set D, and $p\left(\boldsymbol{\theta}\right)$ is the prior distribution of the parameters $\boldsymbol{\theta}$. The posterior distribution can be calculated using Bayes' rule:

$$p\left(\boldsymbol{\theta}|D\right)=\frac{p\left(D|\boldsymbol{\theta}\right)p\left(\boldsymbol{\theta}\right)}{p\left(D\right)}\ , \tag{2}$$

where $p\left(D\right)$ is calculated using the sum rule.

In this paper we will focus on the online version of Bayesian inference, in which the data arrives sequentially, and we do a sequential update of the posterior distribution each time that new data arrives. In each step, the previous posterior distribution is used as the new prior distribution. Therefore, according to Bayes' rule, the posterior distribution at time n is given by:

$$p\left(\boldsymbol{\theta}|D_{n}\right)=\frac{p\left(D_{n}|\boldsymbol{\theta}\right)p\left(\boldsymbol{\theta}|D_{n-1}\right)}{p\left(D_{n}\right)}\ . \tag{3}$$

Unfortunately, calculating the posterior distribution is intractable for most practical probability models. Therefore, we approximate the true posterior using variational methods.

3.1 Online variational Bayes

In variational Bayes, a parametric distribution $q\left(\boldsymbol{\theta}|\phi\right)$ is used for approximating the true posterior distribution $p\left(\boldsymbol{\theta}|\ D\right)$ by minimizing the Kullback-Leibler (KL) divergence with the true posterior distribution.

$$\mathrm{KL}\left(q\left(\boldsymbol{\theta}|\phi\right)||p\left(\boldsymbol{\theta}|D\right)\right)=-\mathbb{E}_{\boldsymbol{\theta}\sim q\left(\boldsymbol{\theta}|\phi\right)}\left[\log\frac{p\left(\boldsymbol{\theta}|D\right)}{q\left(\boldsymbol{\theta}|\phi\right)}\right] \tag{4}$$

The optimal variational parameters are the solution of the following optimization problem:

$$\begin{aligned}\phi^{*}&=\arg\min_{\phi}\int q\left(\boldsymbol{\theta}|\phi\right)\log\frac{q\left(\boldsymbol{\theta}|\phi\right)}{p\left(\boldsymbol{\theta}|D\right)}d\boldsymbol{\theta}=\arg\min_{\phi}\int q\left(\boldsymbol{\theta}|\phi\right)\log\frac{q\left(\boldsymbol{\theta}|\phi\right)}{p\left(D|\boldsymbol{\theta}\right)p\left(\boldsymbol{\theta}\right)}d\boldsymbol{\theta}\\&=\arg\min_{\phi}\mathbb{E}_{\boldsymbol{\theta}\sim q\left(\boldsymbol{\theta}|\phi\right)}\left[\log\left(q\left(\boldsymbol{\theta}|\phi\right)\right)\ -\log\left(p\left(\boldsymbol{\theta}\right)\right)+\mathrm{L}\left(\boldsymbol{\theta}\right)\ \right],\end{aligned} \tag{5}$$

where $\mathrm{L}\left(\boldsymbol{\theta}\right)=-\log\left(p\left(D|\boldsymbol{\theta}\right)\right)$ is the log-likelihood cost function[4].

[4]Notice that we define a cumulative log-likelihood cost function over the data.

In online variational Bayes, we aim to find the posterior in an online setting, where the data arrives sequentially. Similar to Bayesian inference we use the previous approximated posterior as the new prior distribution. For example, at time n the optimal variational parameters are the solution of the following optimization problem:

$$\begin{aligned}\phi^* &= \arg\min_{\phi} \int q_n\left(\boldsymbol{\theta}|\phi\right) \log \frac{q_n\left(\boldsymbol{\theta}|\phi\right)}{p\left(\boldsymbol{\theta}|D_n\right)} d\boldsymbol{\theta} = \arg\min_{\phi} \int q_n\left(\boldsymbol{\theta}|\phi\right) \log \frac{q_n\left(\boldsymbol{\theta}|\phi\right)}{p\left(D_n|\boldsymbol{\theta}\right) q_{n-1}\left(\boldsymbol{\theta}\right)} d\boldsymbol{\theta} \\ &= \arg\min_{\phi} \mathbb{E}_{\boldsymbol{\theta}\sim q_n\left(\boldsymbol{\theta}|\phi\right)} \left[\log\left(q_n\left(\boldsymbol{\theta}|\phi\right)\right) - \log\left(q_{n-1}\left(\boldsymbol{\theta}\right)\right) + \mathrm{L_n}\left(\boldsymbol{\theta}\right)\right], \end{aligned} \tag{6}$$

where $\mathrm{L}_n\left(\boldsymbol{\theta}\right) = -\log\left(p\left(D_n|\boldsymbol{\theta}\right)\right)$ is the log-likelihood cost function.

3.2 Diagonal Gaussian approximation

A standard approach is to define the parametric distribution $q\left(\boldsymbol{\theta}|\phi\right)$ so that all the components of the parameter vector θ would be factorized, i.e. independent ("mean-field approximation"). In addition, in this paper we will focus on the case in which the parametric distribution $q\left(\boldsymbol{\theta}|\phi\right)$ and the prior distribution are Gaussian. Therefore:

$$q_n\left(\boldsymbol{\theta}|\phi\right) = \prod_i \mathcal{N}\left(\theta_i|\mu_i, \sigma_i^2\right), \quad q_{n-1}\left(\boldsymbol{\theta}\right) = \prod_i \mathcal{N}\left(\theta_i|m_i, v_i^2\right) \tag{7}$$

In order to solve the optimization problem in eq. (6), we use the unbiased Monte Carlo gradients, similarly to [4]. We define a deterministic transformation:

$$\theta_i = \mu_i + \varepsilon_i \sigma_i, \quad \varepsilon_i \sim \mathcal{N}\left(0,1\right), \quad \phi = \left(\boldsymbol{\mu}, \boldsymbol{\sigma}\right) \tag{8}$$

The following holds:

$$\frac{\partial}{\partial\phi}\mathbb{E}_{\boldsymbol{\theta}}\left[f\left(\boldsymbol{\theta},\phi\right)\right] = \mathbb{E}_{\varepsilon}\left[\frac{\partial f\left(\boldsymbol{\theta},\phi\right)}{\partial\boldsymbol{\theta}}\frac{\partial\boldsymbol{\theta}}{\partial\phi} + \frac{\partial f\left(\boldsymbol{\theta},\phi\right)}{\partial\phi}\right], \tag{9}$$

where $f\left(\boldsymbol{\theta},\phi\right) = \log\left(q_{n+1}\left(\boldsymbol{\theta}|\phi\right)\right) - \log\left(q_n\left(\boldsymbol{\theta}\right)\right) + \mathrm{L}\left(\boldsymbol{\theta}\right)$. Therefore, we can find a critical point of the objective function by solving the following set of equations:

$$\mathbb{E}_{\varepsilon}\left[\frac{\partial f\left(\boldsymbol{\theta},\phi\right)}{\partial\boldsymbol{\theta}}\frac{\partial\boldsymbol{\theta}}{\partial\phi} + \frac{\partial f\left(\boldsymbol{\theta},\phi\right)}{\partial\phi}\right] = 0\,. \tag{10}$$

Substituting eq. (7) and eq. (8) into eq. (10) we get (see Appendix A for additional details):

$$\mu_i = m_i - v_i^2 \mathbb{E}_{\varepsilon}\left[\frac{\partial \mathrm{L}_n\left(\boldsymbol{\theta}\right)}{\partial\theta_i}\right] \tag{11}$$

$$\sigma_i = v_i\sqrt{1 + \left(\frac{1}{2}v_i\mathbb{E}_{\varepsilon}\left[\frac{\partial \mathrm{L}_n\left(\boldsymbol{\theta}\right)}{\partial\theta_i}\varepsilon_i\right]\right)^2} - \frac{1}{2}v_i^2\mathbb{E}_{\varepsilon}\left[\frac{\partial \mathrm{L}_n\left(\boldsymbol{\theta}\right)}{\partial\theta_i}\varepsilon_i\right] \tag{12}$$

Notice that eq. (11) and eq. (12) are implicit equations, since the derivative $\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial\theta_i}$ is a function of μ_i and σ_i. We approximate the solution using a single explicit iteration of this equation, i.e. evaluate the derivative $\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial\theta_i}$ using the prior parameters. This approximation results in an explicit closed form update rule for μ and σ. This approximation becomes more accurate when we are near the solution of eq. (11). If we are not near the solution, this can lead to a slow rate of convergence. To compensate for this we added a "learning rate" hyper-parameter η to adjust the convergence rate. Also, notice that in the theoretical derivation we assumed that the data arrives sequentially, in an online setting. In practice, however, the log-likelihood cost function does not converge after one epoch since we use a single explicit iteration. Therefore, in Algorithm 1 we repeatedly go over the training set until the convergence criterion is met. In addition, the expectations are approximated using Monte Carlo sampling method (we use K Monte Carlo samples). In terms of computation complexity, those Monte Carlo samples are the major difference from SGD, see Appendix F for further discussion. The full algorithm is described in Algorithm 1.

Algorithm 1 Bayesian Gradient Descent (BGD)

Initialize μ, σ, η, K

Repeat

$$\mu_i \leftarrow \mu_i - \eta\sigma_i^2 \mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\right], \quad \sigma_i \leftarrow \sigma_i \sqrt{1+\left(\frac{1}{2}\sigma_i \mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right]\right)^2} - \frac{1}{2}\sigma_i^2 \mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right]$$

Until convergence criterion is met.

The expectations are estimated using Monte Carlo method, with $\theta_i^{(k)} = \mu_i + \varepsilon_i^{(k)}\sigma_i$:

$$\mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\right] \approx \frac{1}{K}\sum_{k=1}^{K}\frac{\partial \mathrm{L}_n\left(\boldsymbol{\theta}^{(k)}\right)}{\partial \theta_i}, \quad \mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] \approx \frac{1}{K}\sum_{k=1}^{K}\frac{\partial \mathrm{L}_n\left(\boldsymbol{\theta}^{(k)}\right)}{\partial \theta_i}\varepsilon_i^{(k)}$$

3.3 Theoretical properties of Bayesian Gradient Descent

Bayesian Gradient Descent (BGD) consists of a gradient descent algorithm for μ, and a recursive update rule for σ, both result from an approximation to the online Bayes update in eq. (3), hence the name BGD. The learning rate of μ_i is proportional to the uncertainty in the parameter θ_i according to the prior distribution. During the learning process, as more data is seen, the learning rate decreases for parameters with a high degree of certainty, while the learning rate increases for parameters with a high degree of uncertainty. Next, we establish this intuitive idea more precisely.

It is easy to verify that the update rule for σ is a strictly monotonically decreasing function of $\mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right]$. Therefore:

$$\begin{aligned}
\mathrm{E}_\varepsilon\left[\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] > 0 &\Longrightarrow \sigma_i(n) < \sigma_i(n-1) \\
\mathrm{E}_\varepsilon\left[\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] < 0 &\Longrightarrow \sigma_i(n) > \sigma_i(n-1) \\
\mathrm{E}_\varepsilon\left[\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] = 0 &\Longrightarrow \sigma_i(n) = \sigma_i(n-1)
\end{aligned} \tag{13}$$

Next, using a Taylor expansion, we show that for small values of σ, the quantity $\mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right]$, approximates the curvature of the loss:

$$\begin{aligned}
\mathbb{E}_\epsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] &= \mathbb{E}_\varepsilon\left[\left(\frac{\partial \mathrm{L}_n(\boldsymbol{\mu})}{\partial \theta_i} + \sum_j \frac{\partial^2 \mathrm{L}_n(\boldsymbol{\mu})}{\partial \theta_i \partial \theta_j}\epsilon_j\sigma_j + O\left(\|\boldsymbol{\sigma}\|^2\right)\right)\varepsilon_i\right] \\
&= \frac{\partial^2 \mathrm{L}_n(\boldsymbol{\mu})}{\partial^2 \theta_i}\sigma_i + O\left(\|\boldsymbol{\sigma}\|^2\right),
\end{aligned}$$

where we used $\mathbb{E}_\varepsilon[\varepsilon_i] = 0$ and $\mathbb{E}_\varepsilon[\varepsilon_i\varepsilon_j] = \delta_{ij}$ in the last line. Thus, in this case, $\mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right]$ is a finite difference approximation to the component-wise product of the diagonal of the Hessian of the loss, and the vector σ. Therefore, we expect that the uncertainty (learning rate) would decrease in areas with positive curvature (e.g., near local minima), or increase in areas with high negative curvature (e.g., near maxima, or saddles). This seems like a "sensible" behavior of the algorithm, since we wish to converge to local minima, and escape saddles. This is in contrast to many common optimization methods, which are either insensitive to the sign of the curvature, or use it the wrong way [5].

In the case of strongly convex loss, we can make a more rigorous statement, which we prove in Appendix B.

Theorem 1. *We examine BGD with a diagonal Gaussian distribution for $\boldsymbol{\theta}$. If $\mathrm{L}_n(\boldsymbol{\theta})$ is a strongly convex function with parameter $m_n > 0$ and a continuously differentiable function over $\mathbb{R}^n$, then $\mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] \geq \frac{m_n}{\sqrt{2\pi}} > 0$.*

Corollary 1. *If* $\mathrm{L}_n(\boldsymbol{\theta})$ *is strongly convex (concave) function for all* $n \in \mathbb{N}$*, then the sequence* $\{\sigma_i(n)\}_{n=1}^{\infty}$ *is strictly monotonically decreasing (increasing).*

Furthermore, one can generalize these results and show that if a restriction of $\mathrm{L}_n(\boldsymbol{\theta})$ to an axis θ_i is strongly convex (concave) for all $n \in \mathbb{N}$, then $\{\sigma_i(n)\}_{n=1}^{\infty}$ is monotonic decreasing (increasing).

Therefore, in the case of a strongly convex loss function, $\sigma_i = 0$ is the only stable point of eq. (12), which means that we collapse to a point estimation similar to SGD. However, for neural networks σ_i does not generally converge to zero. In this case, the stable point $\sigma_i = 0$ is generally not unique, since $\mathbb{E}_{\varepsilon}\left[\frac{\partial \mathrm{L}(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right]$ implicitly depends on σ_i.

In Appendix G we show the histogram of STD values on MNIST when training for 5000 epochs and demonstrate we do not collapse to a point estimation.

In the case of over-parameterized models and continual learning, only a part of the weights is essential for each task. We hypothesize that if weight θ_i is important to the current task, this implies that, near the minimum, the function $\mathrm{L}_i = \mathrm{L}(\boldsymbol{\theta})|_{(\boldsymbol{\theta})_i=\theta_i}$ is locally convex. Corollary 1 suggests that in this case σ_i would be small. In contrast, the loss will have a flat curvature in the direction of weights which are not important to the task. Therefore, these unimportant weights may have a large uncertainty σ_i. Since BGD introduces the linkage between learning rate and the uncertainty (STD), the training trajectories in the next task would be restricted along the less important weights leading to a good performance on the new task, while retaining the performance on the current task.

4 Experiments

First, in subsection 4.1, we verify that BGD has a good baseline performance on a single tasks. Next, in subsection 4.2, we use several continual learning benchmarks to demonstrate BGD's performance on several consecutive tasks. Notice that BGD is trained under a task-agnostic setting, but compared with various algorithms trained under a task-aware setting. We aimed to provide as a broad variety of comparisons as possible, however, due to the lack of open-source implementations / architecture-specific implementations of other algorithms, not all algorithms are evaluated in all experiments. See Appendix E for further implementation details of all experiments and Appendix D for information about the datasets.

4.1 Classification

As a sanity check, we compare BGD performance on single-task classification to recent Bayesian methods used for classification along with SGD and ADAM.

Table 1: Test accuracy for MNIST classification.

Layer width	BBB (Gaussian)	BGD	SGD
400	98.18%	**98.26%**	98.17%
800	98.01%	**98.33%**	98.16%
1200	97.96%	**98.22%**	98.12%

MNIST classification We compare BGD on MNIST with SGD and BBB [4]. Table 1 summarizes test accuracy[5] of networks with various widths. Overall, BGD performs better than BBB and SGD.

CIFAR10 classification We use CIFAR10 to evaluate BGD's performance on a more complex dataset and a larger network. In this experiment, besides comparison to SGD and ADAM, we compare BGD to a group of recent advanced variational inference algorithms, including both diagonal and non-diagonal methods — K-FAC [21], Noisy K-FAC and Noisy ADAM [36]. For further details about those algorithms, see Appendix C.

[5]Results of SGD and BBB in Table 1 for MNIST are as reported on [4].

Table 2: Classification accuracy on CIFAR10 with modified VGG16 as in [36]. All results except for BGD and ADAM are as reported in [36]. [N/A] values are due to extreme instability.

	Test Accuracy [%]	
Method	**Without BN**	**With BN**
Diagonal methods		
BGD	**88.41**	91.07
SGD	88.35	**91.39**
ADAM	87.12	90.15
BBB [7]	88.31	N/A
NoisyADAM	88.23	N/A
Non-diagonal methods		
KFAC	88.89	**92.13**
NoisyKFAC	**89.35**	92.01

To do so, we followed the experiment described in [36]. There, a VGG16-like[6] [30] architecture was used and data augmentation is applied. We present results both with Batch Normalization [12] and without it in Table 2.

From those classification experiments we conclude that:

1. BGD slightly outperforms BBB.
2. BGD does not experience underfitting and over-pruning, as shown by [32] for BBB. This is true even when training much longer — in Appendix G we show training results on MNIST and CIFAR10 for 5000 epochs.
3. BGD can be applied easily to CIFAR10, as opposed to vanilla BBB.
4. BGD performs only slightly worse than other non-diagonal variational Bayes methods, although it is using only a diagonal approximation to the posterior.

4.2 Continual learning

We compare BGD's performance on three different continual learning experiments conducted in related works - permuted MNIST, CIFAR10/CIFAR100 and a combined vision datasets mix. Surprisingly, BGD performance is comparable to some of the best task-aware algorithms, even though it is unaware of the task changing (i.e., it solves a more challenging problem).

Continual learning on permuted MNIST BGD is compared with diagonal methods on permuted MNIST: "Synaptic Intelligence" (SI), "Variational Continual Learning" (VCL) [35, 25] and SGD as a baseline. Notice that in [35] the author compared SI and "Elastic weight consolidation" (EWC) on permuted MNIST and showed similar performance.

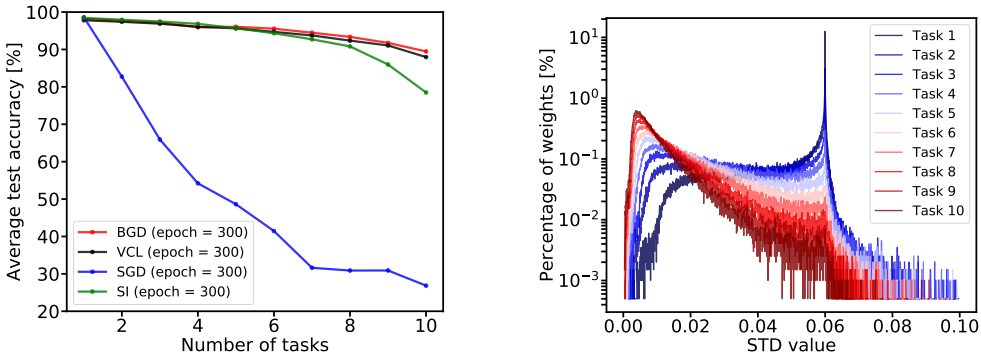
Permuted MNIST is a set of tasks constructed by a random permutation of MNIST pixels. Each task has a different permutation of pixels from the previous one, and the network is trained on the different tasks sequentially. Our setup is two fully connected hidden layers of width 200. We use the same training configuration for all compared algorithms — 300 epochs and batch size of 128.

Average accuracy on seen tasks is reported in Figure 1a. As can be seen, the network learns to solve the tasks with high accuracy even without the knowledge of when the tasks are switched.

Figure 1b shows the histogram of STD values at the end of the training process of each task. The results show that after the first task, a large portion of the weights have STD value which is close to the initial value of 0.06, while a small fraction of them have a much lower value. As training progresses, more weights are assigned with STD values much lower than the initial value, but higher

[6] The detailed network architecture is 32-32-M-64-64-M-128-128-128-M-256-256-256-M-256-256-256-M-FC10, where each number represents the number of filters in a convolutional layer, and M denotes max-pooling - same as [36].

[7] A relaxed version of BBB was used, with $\lambda = 0.1$ as described in [36].



(a) (b)

Figure 1: The left figure shows the average test accuracy on permuted MNIST vs. the number of tasks. BGD (red), VCL (black), SI (green) and SGD (blue). We used mini-batch of size 128 and 300 epochs for all the algorithms. Notice that, in contrast to VCL and SI, BGD is task-agnostic (i.e. unaware of tasks changing), while still significantly alleviates catastrophic forgetting. The right figure shows the histogram of STD values at the end of the training process of each task, and the initial STD value is 0.06.

by at least a factor of 10 then the minimal value of σ, which is $\sim 10^{-4}$. These results support our hypothesis in subsection 3.3 that only a small part of the weights is essential for each task, and as training progresses percentage of weights with small STD increases as more tasks are seen. It also shows that we do not collapse to point estimation.

See Appendix H for results on additional configurations and more details on the experiment's setting.

Continual learning on CIFAR10/CIFAR100 We follow SI [35] experiment on CIFAR10/CIFAR100 — training sequentially on six tasks from CIFAR10 and CIFAR100. The first task is the full CIFAR10 dataset, and the next five tasks are subsets of CIFAR100 with ten classes, where we train each task for 150 epochs. As a reference, we present results of SGD and SI. We used the same network as in the original experiment, which consists of four convolutional layers followed by a fully connected layer with dropout shared by all six tasks. Since we follow the original setup of this experiment which uses separated last fully connected layer per task, representation learning is task-agnostic but the last layer is task specific.

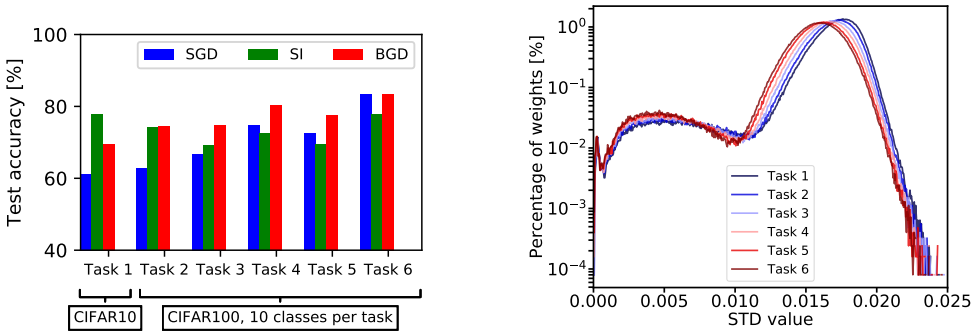
The results (Figure 2a) show that the network is able to learn relevant representations (the convolutional layers) even without the knowledge of when the tasks are switched, while attaining a good balance between retaining reasonable accuracy on previous tasks and achieving high accuracy on newer ones.

Figure 2b shows the histogram of STD values at the end of the training process of each task. Similar to the results for permuted MNIST, as more tasks are seen the percentage of weights with STD values smaller than initial STD value (of 0.019) increases (the minimal value of σ is $\sim 10^{-4}$).

Continual learning on vision datasets Finally, we followed [27] and challenged our algorithm with the vision datasets experiment. In this experiment, we train sequentially on MNIST, notMNIST [8], FashionMNIST, SVHN and CIFAR10 [16, 34, 24, 15]. Training is done in a sequential way with 20 epochs per task — in epochs 1-20 we train on MNIST (first task), and on epochs 81-100 we train on CIFAR10 (last task). All five datasets consist of about 50,000 training images from 10 different classes, but they differ from each other in various ways: black and white vs. RGB, letters and digits vs. vehicles and animals etc. See Appendix D for further details about the datasets. We use the exact same setup as in [27] for the comparison — LeNet-like [16] architecture with separated last layer for each task as in CIFAR10/CIFAR100 experiment.

As seen in the results (Table 3), BGD achieves decent average accuracy, despite being task-agnostic.

[8] Originally published at http://yaroslavvb.blogspot.co.uk/2011/09/notmnist-dataset.html and downloaded from https://github.com/davidflanagan/notMNIST-to-MNIST



(a) (b)

Figure 2: The left figure shows the test accuracy per task on last epoch for continual learning on CIFAR10 and subsets of CIFAR100. Task 1 is the first seen task and task six is the last. The right figure shows the histogram of STD values at the end of the training process of each task, and the initial STD value is 0.019.

Table 3: Accuracy for each task after training sequentially on all tasks. BGD provides competitive results, while being the only task-agnostic algorithm. PTL stands for Per-Task Laplace (one penalty per task), AL is Approximate Laplace (Laplace approximation of the full posterior at the mode of the approximate objective) and OL is Online Laplace approximation. Results for SI, PTL, AL and OL are as reported in [27]

		Test accuracy [%] on the end of last task (CIFAR10)					
Method	**Task-agnostic**	**Average**	**MNIST**	**notMNIST**	**F-MNIST**	**SVHN**	**CIFAR10**
Diagonal methods							
BGD	✓	81.37	86.42	89.23	83.05	**82.21**	**65.96**
SGD	✓	69.64	84.79	82.12	65.91	52.31	63.08
ADAM	✓	29.67	17.39	26.26	25.02	15.10	64.62
SI	✗	77.21	87.27	79.12	84.61	77.44	57.61
PTL	✗	**82.96**	**97.83**	**94.73**	89.13	79.80	53.29
AL	✗	82.55	96.56	92.33	**89.27**	78.00	56.57
OL	✗	82.71	96.48	93.41	88.09	81.79	53.80
Non-Diagonal methods							
PTL	✗	85.32	97.85	**94.92**	89.31	**85.75**	58.78
AL	✗	85.35	**97.90**	94.88	90.08	85.24	58.63
OL	✗	**85.40**	97.17	94.78	**90.36**	85.59	**59.11**

5 Discussion and future work

In this work, we present for the first time an efficient approach for task-agnostic continual learning. It mitigates the notorious catastrophic forgetting phenomena that plague neural networks, without being aware of the change in task. This important property can allow future models to better adapt to new tasks without explicitly instructed to do so, enabling them to learn in realistic continual learning settings.

Our method, Bayesian Gradient Descent (BGD) show competitive results with various state-of-the-art continual learning algorithms while being unaware of the task-switch. It relies on solid theoretical foundations, and can be applied easily to advanced DNN architectures, including convolution layers and Batch-Normalization, and works on a broad variety of datasets.

Besides being a continual learning method, BGD had better classification accuracy than previous Bayesian methods on MNIST, and was on par with SGD on CIFAR10 (without Batch-Normalization). Despite being an online version of the variational Bayes approach of [4], it does not seem to have the underfitting and over-pruning issues previously observed in the variational Bayes approach of [32]. This allowed us to scale this approach beyond MNIST, to CIFAR10. The BGD approach is

also closely related to the assumed density filtering approach of [31, 9]. However, these methods rely on certain analytic approximations which are not easily applicable to different neural architectures (e.g. convnets). In contrast, it is straightforward to implement BGD for any neural architecture. This approach is also somewhat similar to the Stochastic Gradient Langevin Dynamics (SGLD) approach [33, 3], in the sense that we use multiple copies of the network during training. However, in contrast to the SGLD approach, we are not required to store all copies of the networks for inference ([3] had to use distillation to overcome this), but only two parameters for each weight (μ and σ).

There are many possible extensions and uses of BGD, which were not explored in this work. First, in this work, our variational approximation used a diagonal Gaussian distribution. This assumption may be relaxed in the future to non-diagonal or non-Gaussian (e.g., mixtures) distributions, to allow better flexibility during learning. Second, this work focuses on catastrophic forgetting, however, Bayesian neural networks might also be beneficial for many other uses. For example, it enables better weight pruning (See Appendix I); uncertainty estimates over the network output; selection of hyper-parameters and models in a principled framework; and guided data collection (active learning).

Acknowledgments

The authors are grateful to R. Amit, M. Shpigel Nacson, N. Merlis and O. Rabinovich for helpful comments on the manuscript. This research was supported by the Israel Science foundation (grant No. 31/1031), and by the Taub foundation. A Titan Xp used for this research was donated by the NVIDIA Corporation.

References

[1] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2): 251–276, 1998.

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

[3] Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*, pages 3438–3446, 2015.

[4] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622, 2015.

[5] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *NIPS*, pages 1–9, 2014. ISSN 10495258.

[6] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[10] José Miguel Hernández-Lobato, Yingzhen Li, Mark Rowland, Daniel Hernández-Lobato, Thang Bui, and Richard Eric Turner. Black-box α-divergence minimization. 2016.

[11] G E Hinton and D Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT '93*, 1993. URL `http://dl.acm.org/citation.cfm?id=168306`.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[14] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[15] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] Sang-Woo Lee, Jin-Hwa Kim, JungWoo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. 2017.

[18] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[19] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[20] D J C MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 472(1):448–472, 1992.

[21] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[22] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[23] R M Neal. *Bayesian Learning for Neural Networks*. PhD thesis, Dept. of Computer Science, University of Toronto, 1994.

[24] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[25] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[26] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

[27] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. *arXiv preprint arXiv:1805.07810*, 2018.

[28] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[29] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971, 2014.

[32] Brian Trippe and Richard Turner. Overpruning in variational bayesian neural networks. *arXiv preprint arXiv:1801.06230*, 2018.

[33] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

[34] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[35] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995, 2017.

[36] Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. *arXiv preprint arXiv:1712.02390*, 2017.

Appendix

A Derivation of eqs. (11) and (12)

In this section we provide additional details on the derivation of eq. (11) and eq. (12). The objective function is $f(\boldsymbol{\theta}, \phi) = \log(q_n(\boldsymbol{\theta}|\phi)) - \log(q_{n-1}(\boldsymbol{\theta})) + \mathrm{L}_n(\boldsymbol{\theta})$, where:

$$\log(q_n(\boldsymbol{\theta}|\phi)) = -\frac{N}{2}\log(2\pi) - \sum_k \log(\sigma_k) - \sum_k \frac{1}{2\sigma_k^2}(\theta_k - \mu_k)^2, \tag{14}$$

$$\log(q_{n-1}(\boldsymbol{\theta})) = -\frac{N}{2}\log(2\pi) - \sum_k \log(v_k) - \sum_k \frac{1}{2v_k^2}(\theta_k - m_k)^2. \tag{15}$$

We derive eq. (11) by using the first-order necessary conditions for the optimal μ_i:

$$\mathbb{E}_\varepsilon\left[\frac{\partial f(\boldsymbol{\theta}, \phi)}{\partial \theta_i}\frac{\partial \theta_i}{\partial \mu_i} + \frac{\partial f(\boldsymbol{\theta}, \phi)}{\partial \mu_i}\right] = 0 \tag{16}$$

Substituting the derivatives, we obtain:

$$\begin{aligned}&\mathbb{E}_\varepsilon\Big[-\frac{1}{\sigma_i^2}(\theta_i - \mu_i) + \frac{1}{v_i^2}(\theta_i - m_i) + \frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i} + \frac{1}{\sigma_i^2}(\theta_i - \mu_i)\Big] \\ &= \frac{1}{v_i^2}(\mu_i - m_i) + E_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\right] = 0.\end{aligned} \tag{17}$$

And so we obtained eq. (11).

Next, we derive eq. (12), using the first-order necessary conditions for optimal σ_i:

$$\mathbb{E}_\varepsilon\left[\frac{\partial f(\boldsymbol{\theta}, \phi)}{\partial \theta_i}\frac{\partial \theta_i}{\partial \sigma_i} + \frac{\partial f(\boldsymbol{\theta}, \phi)}{\partial \sigma_i}\right] = 0. \tag{18}$$

Substituting the derivatives we obtain:

$$\begin{aligned}&\mathbb{E}_\varepsilon\left[\left(-\frac{1}{\sigma_i^2}(\theta_i - \mu_i) + \frac{1}{v_i^2}(\theta_i - m_i) + \frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\right)\varepsilon_i - \frac{1}{\sigma_i} + \frac{1}{\sigma_i^3}(\theta_i - \mu_i)^2\right] \\ &= -\frac{1}{\sigma_i} + \frac{\sigma_i}{v_i^2} + \mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] = 0.\end{aligned} \tag{19}$$

We get a quadratic equation for σ_i

$$\sigma_i^2 + \sigma_i v_i^2 E_{q(\varepsilon)}\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] - v_i^2 = 0. \tag{20}$$

Since $\sigma_i > 0$, the solution is eq. (12).

B Proof of Theorem 1

Proof. We define $\theta_j = \mu_j + \varepsilon_j \sigma_j$ where $\varepsilon_j \sim \mathcal{N}(0, 1)$. According to the smoothing theorem, the following holds

$$\mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\right] = \mathbb{E}_{\varepsilon_{j\neq i}}\left[\mathbb{E}_{\varepsilon_i}\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\middle|\,\varepsilon_{j\neq i}\right]\right]. \tag{21}$$

The conditional expectation is:

$$\mathbb{E}_{\varepsilon_i}\left[\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i\middle|\,\varepsilon_{j\neq i}\right] = \int_{-\infty}^{\infty}\frac{\partial \mathrm{L}_n(\boldsymbol{\theta})}{\partial \theta_i}\varepsilon_i f_{\varepsilon_i}(\varepsilon_i)\,d\varepsilon_i, \tag{22}$$

where f_{ε_i} is the probability density function of a standard normal distribution. Since f_{ε_i} is an even function

$$\begin{aligned}
&\mathrm{E}_{\varepsilon_i}\left[\frac{\partial \mathrm{L}_n\left(\boldsymbol{\theta}\right)}{\partial \theta_i}\varepsilon_i \middle| \varepsilon_{j\neq i}\right] \\
&\quad = \int_0^\infty \frac{\partial \mathrm{L}_n\left(\mu_i + \varepsilon_i\sigma_i, \theta_{-i}\right)}{\partial \theta_i}\varepsilon_i f_{\varepsilon_i}\left(\varepsilon_i\right) d\varepsilon_i \\
&\qquad - \int_0^\infty \frac{\partial \mathrm{L}_n\left(\mu_i - \varepsilon_i\sigma_i, \theta_{-i}\right)}{\partial \theta_i}\varepsilon_i f_{\varepsilon_i}\left(\varepsilon_i\right) d\varepsilon_i.
\end{aligned} \tag{23}$$

Now, since $\mathrm{L}_n\left(\boldsymbol{\theta}\right)$ is strongly convex function with parameter $m_n > 0$ and continuously differentiable function over $\mathbb{R}^d$, the following holds $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^d$:

$$\left(\nabla \mathrm{L}_n\left(\boldsymbol{\theta}_1\right) - \nabla \mathrm{L}_n\left(\boldsymbol{\theta}_2\right)\right)^T \left(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\right) \geq m_n \left\|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\right\|. \tag{24}$$

For $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ such that

$$\left(\boldsymbol{\theta}_1\right)_j = \begin{cases} \left(\boldsymbol{\theta}_2\right)_j, & j \neq i \\ \mu_i + \varepsilon_i\sigma_i, & j = i, \end{cases} \tag{25}$$

$$\left(\boldsymbol{\theta}_2\right)_j = \begin{cases} \left(\boldsymbol{\theta}_1\right)_j, & j \neq i \\ \mu_i - \varepsilon_i\sigma_i, & j = i, \end{cases} \tag{26}$$

the following holds:

$$\left(\frac{\partial \mathrm{L}\left(\boldsymbol{\theta}_1\right)}{\partial \theta_i} - \frac{\partial \mathrm{L}\left(\boldsymbol{\theta}_2\right)}{\partial \theta_i}\right)\varepsilon_i \geq m_n \left|\varepsilon_i\right|. \tag{27}$$

Therefore, substituting this inequality into eq. (17), we obtain:

$$\mathbb{E}_\varepsilon\left[\frac{\partial \mathrm{L}\left(\boldsymbol{\theta}\right)}{\partial \theta_i}\varepsilon_i\right] \geq \frac{m_n}{\sqrt{2\pi}} > 0. \tag{28}$$

□

C Additional related work

Bayesian inference for neural networks has been a subject of significant interest over many years. As exact Bayesian inference is intractable (for any realistic network size), much research has been focused on approximation techniques. Most modern techniques stemmed from previous seminal works which used either a Laplace approximation [20], variational methods [11], or Monte Carlo methods [23]. In the last years, many methods for approximating the posterior distribution have been suggested, falling into one of these categories. Those methods include assumed density filtering [31, 9], approximate power Expectation Propagation [10], Stochastic Langevin Gradient Descent [33, 3], incremental moment matching [17] and variational Bayes [6, 4].

In this work, we will focus on variational Bayes methods. Practical variational Bayes for modern neural networks was first introduced by [6], where parametric distribution is used to approximate the posterior distribution by minimizing the variational free energy. Calculating the variational free energy is intractable for general neural networks, and thus [6] estimated its gradients using a biased Monte Carlo method, and used stochastic gradient descent (SGD) to perform minimization. In a later work, [4] used a re-parameterization trick to introduce an unbiased estimator for the gradients. Variational Bayes methods were also used extensively on various probabilistic models including recurrent neural networks [6], auto-encoder [13] and fully connected networks [4]. [21] and [36] suggested using the connection between natural gradient descent [1] and variational inference to perform natural gradient optimization in deep neural networks.

D Datasets

MNIST is a database of handwritten digits, which has a training set of 60,000 examples, and a test set of 10,000 examples — each a 28×28 image. The image is labeled with a number in the range 0 to 9.

Permuted MNIST is a set of tasks constructed by a random permutation of MNIST pixels. Each task has a different permutation of pixels from the previous one.

CIFAR10 is a dataset which consists of 60000 32×32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

CIFAR100 is a dataset which consists of 60000 32×32 color images in 100 classes, with 600 images per class [15]. There are 50000 training images and 10000 test images. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

FashionMNIST is a dataset comprising of 28×28 grayscale images of 70,000 fashion products from 10 categories. The training set has 60,000 images and the test set has 10,000 images.

notMNIST is a dataset with 10 classes, with letters A-J taken from different fonts. It has similar characteristics like MNIST - same pixel size (28×28), grayscale images and comprised of 10-class.

SVHN is a real-world image dataset with 10 classes, 1 for each digit. It has RGB images of size 32×32 obtained from house numbers in Google Street View images.

E Implementation details

We initialize the mean of the weights μ by sampling from a Gaussian distribution with a zero mean and a variance of $2/(n_{\text{input}} + n_{\text{output}})$ as in [7]. We use 10 Monte Carlo samples to estimate the expected gradient during training, and average the accuracy of 10 sampled networks during testing, unless stated otherwise.

MNIST classification We use a fully connected neural network with two hidden layers of various widths, ReLU's as activation functions and softmax output layer with 10 units. We preprocessed the data to have pixel values in the range of $[0, 1]$. We use mini-batch of size 128 and $\eta = 1$. In addition, we used a validation set to find the initialization value for σ. In order to compare the results of Bayesian Gradient Descent (BGD) with the results of Bayes By Backprop (BBB) algorithm by [4], we used a training set of 50,000 examples and validation set of 10,000 examples and present results of BBB with the same prior as BGD used. We train the network for 600 epochs.

CIFAR10 classification SGD was trained with learning rate starting at 0.01 and divided by 10 every 100 epochs, momentum was set to 0.9. On both BGD and SGD we used data augmentation of random cropping and flipping. Both experiments use a batch size of 128 and 10 Monte Carlo iterations. In the experiment with Batch-Normalization (BN) initial STD is 0.011, η set to 8. μ is initialized using PyTorch 0.3.1 for convolution layers, bias is initialized to 0 and μ of BN layers scaling and shifting parameters are initialized to 1 and 0. In the experiment without Batch-Normalization (BN) initial std is 0.015, η set to 10. μ of convolution layers is initialized using He initialization.

On BGD the initial STD was set to 0.015 and $\eta = 6$.

Permutd MNIST We use a fully connected neural network with 2 hidden layers of 200 width, ReLUs as activation functions and softmax output layer with 10 units. We trained the network using Bayesian Gradient Descent. The preprocessing is the same as with MNIST classification, but we used a training set of 60,000 examples. BGD was trained with mini-batch of size 128 and $\eta = 1$ for 300 epochs.

CIFAR10/CIFAR100 continual learning Batch size is 256. On BGD we used MAP for inference method, initial STD was 0.019 and $\eta = 2$. For SGD we used constant learning rate of 0.01 and for SI we used $c = 0.01$. Hyper parameters for all algorithms were chosen after using grid search and taking the optimal value.

Table 4: Average runtime of a single training epoch with different numbers of Monte Carlo samples

MC iterations	**Runtime** [seconds]	**Vs. SGD**
SGD	7.8	$\times 1$
2 (BGD)	18.9	$\times$ 2.4
4 (BGD)	35.2	$\times$ 4.5
10 (BGD)	83.8	$\times$ 10.7

Vision datasets mix We followed the experiment as described in [27]. We use a batch size of 64, and we normalize the datasets to have zero mean and unit variance. The network architecture is LeNet like with 2 convolution layers with 20 and 50 channels and kernel size of 5, each convolution layer is followed by a Relu activations function and max pool, and the two layers are followed with fully connected layer of size 500 before the last layer.

SGD baseline was trained with a constant learning rate of 0.001 and ADAM used eps $1e-08$, LR of 0.001 and betas: (0.9, 0.999). BGD trained with initial STD of 0.02, η set to 1 and batch size 64.

Weight pruning on MNIST We train a fully connected network with two hidden layers of 1200 width.

F Complexity

BGD requires $\times 2$ more parameters compared to SGD, as it stores both the mean and the STD per weight. In terms of time complexity, the major difference between SGD and BGD arises from the estimation of the expected gradients using Monte Carlo samples during training. Since those Monte Carlo samples are completely independent the algorithm is embarrassingly parallel.

Specifically, given a mini-batch: for each Monte Carlo sample, BGD generates a random network using mu and sigma, then making a forward-backward pass with the randomized weights.

Two main implementation methods are available (using 10 Monte Carlo samples as an example):

1. Producing the (10) Monte Carlo samples sequentially, thus saving only a single randomized network in memory at a time (decreasing memory usage, increasing runtime).
2. Producing the (10) Monte Carlo samples in parallel, thus saving (10) randomized networks in memory (increasing memory usage, decreasing runtime).

We analyzed how the number of Monte Carlo iterations affects the runtime on CIFAR-10 using the first method of implementation (sequential MC samples). The results, reported in Table 4, show that runtime is indeed a linear function of the number of MC iterations. In the experiments we used a single GPU (GeForce GTX 1080 Ti).

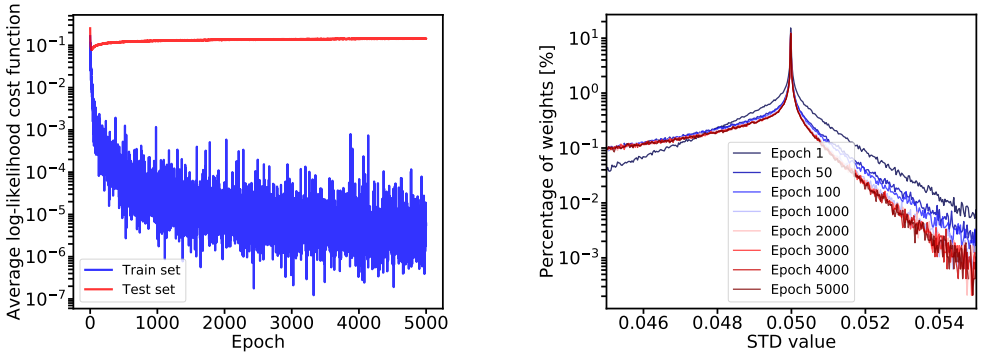
G 5000 epochs training

MNIST We use the MNIST classification experiment to demonstrate the convergence of the log-likelihood cost function and the histogram of STD values. We train a fully connected neural network with two hidden layers and layer width of 400 for 5000 epochs.

Figure 3a shows the log-likelihood cost function of the training set and the test set. As can be seen, the log-likelihood cost function on the training set decreases during the training process and converges to a low value. Thus, BGD does not experience underfitting and over-pruning as was shown by [32] for BBB.

Figure 3b shows the histogram of STD values during the training process. As can be seen, the histogram of STD values converges. This demonstrates that σ_i does not collapse to zero even after 5000 epochs.

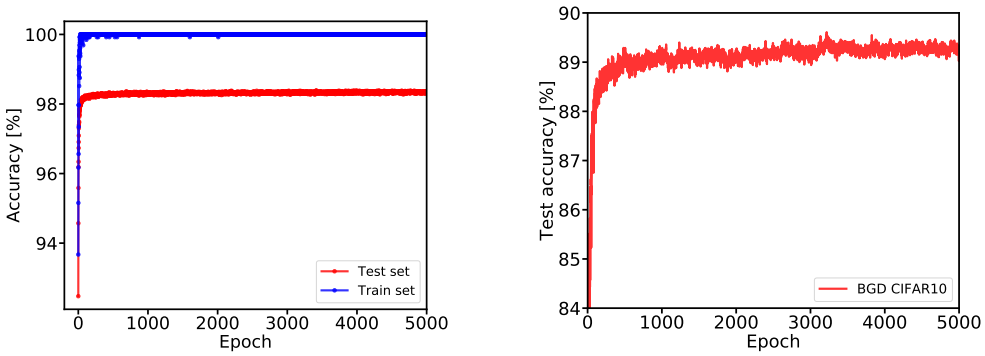
Figure 4a shows the learning curve of the train set and the test set. As can be seen, the test accuracy does not drop even if we continue to train for 5000 epochs.



(a) (b)

Figure 3: The left figure shows the average log-likelihood cost function of the train set and the test set - layer width 400. The right figure shows the histogram of STD values, the initial STD value is 0.05.

CIFAR10 To further show that BGD does not experience underfitting and over-pruning, we trained VGG11 [9]. Figure 4b show the test accuracy during the 5000 epochs.



(a) (b)

Figure 4: The left figure shows test the accuracy and train accuracy on MNIST dataset- layer width 400. The right figure shows the test accuracy on CIFAR10 using VGG11.

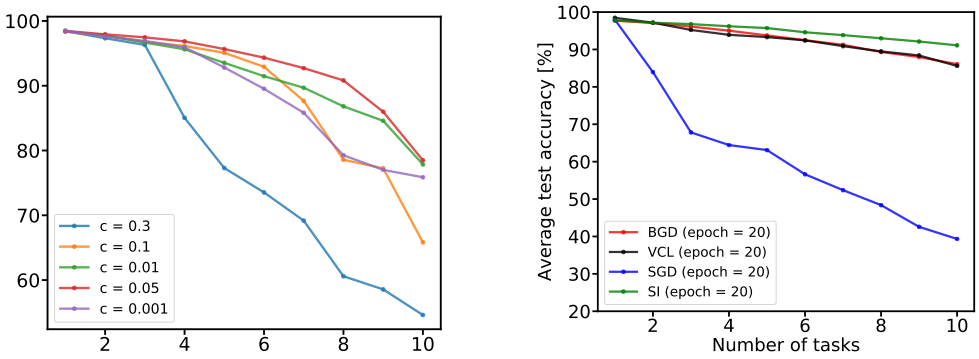
H Permuted MNIST experiment

In this experiment we use a fully connected neural network with 2 hidden layers of 200 width, ReLUs as activation functions and softmax output layer with 10 units. We use the same training configuration for all the algorithms — 300 epochs and batch size of 128 (BGD was trained with $\eta = 1$). We run an hyper-parameters tuning for SI using $c = (0.3, 0.1, 0.05, 0.01, 0.001)$ and select the best one (c = 0.05) as a baseline (See Figure 5a).

We ran an additional simulation with the same architecture on exactly the configuration as in [35], see Figure 5b

Moreover, we ran a simulation with a larger architecture using a fully connected neural network with 2 hidden layers of 2000 width. But, in this simulation we run SI with on exactly the configuration as in [35] (20 epochs and batch size of 256) and BGD with 300 epochs and batch size of 128. we were not able to run VCL code on this large architecture, see Figure 6.

[9] We trained a full VGG11. The detailed network architecture is 64-M-128-M-256-256-M-512-512-M-512-512-M-FC10



(a) (b)

Figure 5: The left figure SI average test accuracy on permuted MNIST vs. the number of tasks with different hyper-parameter values. The right figure shows the average test accuracy on permuted MNIST vs. the number of tasks. BGD (red), VCL (black), SI (green) and SGD (blue), We used mini-batch of size 256 and 20 epochs for all the algorithms.

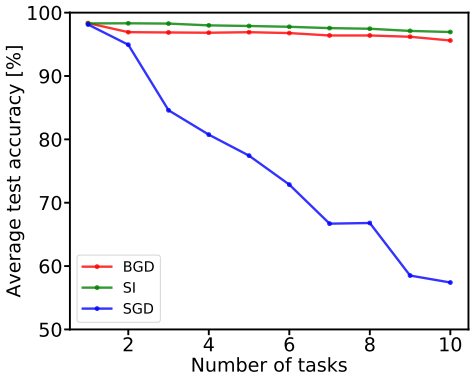


Figure 6: The average test accuracy on permuted MNIST vs. the number of tasks. BGD (red), SI (green) and SGD (blue).

I Weight pruning

STD and mean learned by BGD can be used for weights pruning by defining the signal to noise ratio (SNR) as $\mathrm{SNR} = |\mu| / \sigma$ and using it as a measure of a weight's necessity [6, 4]. For large STD the sampled weight value is expected to spread over a large range of values. For small mean, the sampled value is likely to be close to zero. In both cases, the weight might have no effect on the network's output or even a negative effect in terms of loss. Hence, the SNR seems to be a relevant measure for weights pruning, preferring to prune weights with lower SNR. When comparing with SGD, we prune weights with low absolute value. When comparing with SGD, we used the absolute value as a measure — pruning weights with low absolute value.

Weight pruning on MNIST Table 5 summarizes the results of weight pruning on MNIST. We train a fully connected network with two hidden layers of 1200 width. Results are compared to SGD and BBB. Note that in BBB the results for pruning were reported for a mixture of Gaussians, (both prior and variational approximation) which is a more powerful than a single Gaussian used by BGD — this improved the baseline accuracy results, but had similar relative success in weight pruning as our method.

Note that our goal was not to improve the current performance of BBB (e.g. weight pruning on MNIST), but rather to develop a Bayesian algorithm that could be easily scaled while performing similarly. The next experiment demonstrates weight pruning on CIFAR10.

Table 5: Weight pruning on MNIST. Showing the ratio Error(Pruning)/Error(No Pruning), smaller values are better.

Dataset	% Removed (# Weights)	Error ratio BGD	SGD	BBB
MNIST	0% (2.4M)	1	1	1
	95% (120K)	1.05	4.28	**1.04**
CIFAR10	0% (9.2M)	1	1	✗
	90% (0.9M)	**1.04**	7.7	✗

Weight pruning on CIFAR10 We evaluated a pruned version of VGG11 trained by BGD, pruning 90% of the weights. BGD pruned network succeed to retain most of the accuracy achieved by the non-pruned network, while SGD pruned networks accuracy drops. Results are reported in Table 5. BBB results were not tested on CIFAR10 in [4]. In addition, we were not able to run BBB on CIFAR10 with VGG11. The results indicate that using BGD we are able to easily prune weights.

Figure 7 shows the test accuracy on CIFAR10 with VGG11 as a function of pruning percentage.

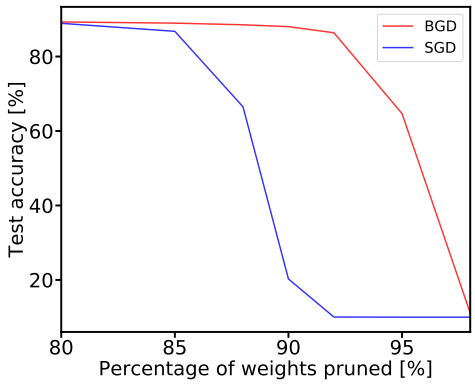


Figure 7: Test accuracy as a function of pruning percentage for CIFAR10 with VGG11.