Training Bayesian Neural Network with Various Ensemble Learning Strategies

Student Name: Chao Wang

Supervisor Name: Sebastian Schmon

Submitted as part of the degree of M.Sc. MISCADA to the

Board of Examiners in the Department of Computer Science, Durham University.

Abstract –

Context / Background: The laboratory achievements related to artificial neural networks (ANNs) have experienced explosive growth in the last decade. Despite the huge success that ANNs have made in both academia and industry, it is a critical issue existing in traditional ANNs that they are nearly all trained using frequentist schemes and then present their results in the form of a point estimate, meaning that they are inclined to be over-confident about their predictions. This shortcoming impedes the further deployment of ANNs in some applications which cannot afford the price of incorrect predictions from ANNs. To address this concern, a variety of training techniques are developed to improve the performance of ANNs. Besides, it is necessary to provide an approach to quantify the uncertainty associated with predictions.

Aims: This dissertation aims to explore the potential of Bayesian neural networks (BNNs) in generalization performance enhancement and uncertainty estimation. Besides, some novel ensemble learning strategies are exploited for training BNNs, and their contributions to improving the performance of BNNs are compared.

Method: Firstly, a multi-layer perceptron network with one fully connected hidden layer is built and trained on the MNIST dataset using different methods, including classical backpropagation, Hamiltonian Monte Carlo (HMC), variational inference (VI), classical ensemble, and variational inference ensemble. Secondly, for a Bayesian convolutional neural network (BCNN) trained using VI on the Fashion-MNIST dataset, various ensembling strategies are applied to its training algorithm, including traditional ensembling, Snapshot Ensembling, Fast Geometric Ensembling (FGE), and Stochastic Weight Averaging (SWA). The generalization performance and training cost of all the above training algorithms are evaluated and compared.

Results: (1) BNNs achieve better generalization performance than that of traditional point estimate neural networks. (2) BNNs can express their uncertainty about predictions through histograms of predictions. (3) Compared to VI, HMC makes BNNs have better prediction accuracy but consumes more training time. (4) Ensembling strategies are helpful to improve the generalization performance of BNNs. (5) SWA can improve the generalization performance of BNNs without much additional training time consumption.

Conclusions: BNNs can not only have great generalization performance but also provide uncertainty estimates associated with their predictions. Ensembling techniques, which are usually used to improve the performance of traditional point estimate neural networks, can be employed to train BNNs as well.

Keywords **–** Bayesian neural network; Hamiltonian Monte Carlo; variational inference; Fast Geometric Ensembling; Stochastic Weight Averaging

I. INTRODUCTION

A. Background

In human history, many ground-breaking scientific and technological innovations are caused by the contribution of biomimicry. As a representative example of biomimicry in machine learning, the inspiration for ANNs comes from observing the working mechanism of neural networks in the human brain. For instance, LeNet-5 was proposed by LeCun *et al.* (1989) achieving an incredible accuracy (nearly 99%) on hand-written digit image classification, then it was immediately deployed in ATMs to recognize digits on checks. This is

the first commercial success case of ANNs. Nevertheless, the development of ANNs experienced a bottleneck in the subsequent twenty years due to technical issues including a shortage of big training datasets and sufficient computing resources. This stagnant period did not end until AlexNet raised prediction accuracy by approximately 10% in the ImageNet contest (Krizhevsky *et al.*, 2012). Afterwards, more and more ANNs with complex architecture were created and demonstrated great performance. Nowadays, some advanced ANNs have been able to provide state-of-the-art solutions for machine learning problems to improve the quality of our living standards.

In the majority of cases, ANNs are usually designed and trained in a frequentist way. For an available dataset, researchers define the architecture of ANNs and loss function, then optimize network parameters (*i.e.*, weights and biases, both of which will be called weights and denoted as w in the following part of this dissertation). If we look at this process from the probabilistic perspective, there will be two main approaches for ANNs learning weights from the dataset. The first approach is maximum likelihood estimate (MLE), which means, given observed data D, weights w^{MLE} can be estimated by maximizing the likelihood function:

$$w^{MLE} = arg\ max_w\ logP(D|w). \tag{1}$$

This is what the training algorithms of traditional ANNs do using classical backpropagation and gradient descent. The other approach is maximum a posteriori (MAP), which means, given observed data D, weights w^{MAP} can be estimated by maximizing the posterior function:

$$\begin{aligned} w^{MAP} &= argmax_w logP(w|D) \\ &= arg\ max_w\ logP(D|w) + logP(w). \end{aligned} \tag{2}$$

The main difference between these two approaches is the prior term $P(w)$ which yields regularization. MAP is equivalent to MLE with regularization techniques. Therefore, the network weights and predictions of traditional ANNs trained in this frequentist mode are both presented as single fixed values. These traditional ANNs are often called point estimate neural networks.

B. Research Question

However, fatal flaws are existing in point estimate neural networks. Firstly, although point estimate neural networks with deep architecture are capable to represent functions with great non-linearities and provide state-of-the-art solutions for complex machine learning problems, the increasing layers make networks prone to overfitting (Goan and Fookes, 2020). Secondly, there is a concern among people about how users are supposed to deal with situations where ANNs output unreliable results without any warning. If a human being is asked a question related to a subject they never studied, the human being will not give a specific answer but 'I don't know'. In real life, it is very common for human beings to express uncertainty and ignorance about their decisions. However, most of point estimate neural networks mentioned above do not have such seemingly low-level ability. For example, if an image of English letters is inputted into a hand-written digit classifier of ANNs, the model will assign a digit label for this image. In other words, traditional ANNs don't know how to deal with input data that is unrelated to the dataset they trained on. This flaw limits the application of ANNs in products like self-driving cars, medical equipment, etc. Therefore, it is very necessary for ANNs to provide not only accurate predictions but also corresponding uncertainty estimates.

C. Methodology

To address these issues, more recent research puts forward the idea of Bayesian neural network (BNN). BNNs are defined as stochastic ANNs trained using Bayesian Inference (*e.g.*, Jospin *et al.*, 2022). Different from traditional point estimate neural networks, the weights of BNNs are drawn from probability distributions rather than single fixed values (see Figure 1). Bayes' theorem acts as a powerful tool to infer the probability distributions of network weights. Bayes' theorem states that:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} = \frac{P(D,\theta)}{\int_\theta P(D,\theta')\,d\theta'}. \tag{3}$$

θ is a parameter, and D is data that will update θ. The probability distribution $P(D|\theta)$ is called the likelihood.

$P(\theta)$ is the prior. $P(D) = \int_{\theta} P(D, \theta')d\theta'$ is called the evidence or the marginal likelihood. $P(\theta|D)$ is the posterior. From the Bayesian perspective, probability is a belief in the occurrence of events rather than the frequency, and the prior belief is an important influencing factor for the posterior belief. Therefore, the Bayes' formula (3) can be understood as that the prior belief $P(\theta)$ is updated to the posterior belief $P(\theta|D)$ after learning D. To adapt Bayes' formula for BNNs, θ is considered as the network weights w that we are going to optimize, and $D = (x, y)$, where x and y are inputs and labels, respectively. Hence, Bayes' formula can also be presented as:

$$P(w|D) = P(w|x, y) = \frac{P(y|x, w)P(w)}{\int_w P(y|x, w')P(w')dw'} \propto P(y|x, w)P(w). \quad (4)$$

$P(w|D)$, namely $P(w|x, y)$, is the posterior distribution of network weights w learned from the dataset. In fact, as the evidence term $\int_w P(y|x, w')P(w')dw'$ in this formula is extremely difficult to compute, there are two common methods to approximate the posterior: (1) Markov chain Monte Carlo; (2) variational inference. Both of them will be introduced in detail in the following sections of this dissertation. Next, if BNNs are used to do predictions based on inputs $\hat{x}$, their outputs $\hat{y}$ will also be presented in the form of probability distribution.

$$P(\hat{y}|\hat{x}, D) = \mathbb{E}_{P(w|D)}[P(\hat{y}|\hat{x}, w)] = \int_w P(\hat{y}|\hat{x}, w')P(w'|D)dw'. \quad (5)$$

The probability distribution of outputs $P(\hat{y}|\hat{x}, D)$ is called the posterior predictive. In most cases, the result of the posterior predictive is approximated through a Monte Carlo approach.

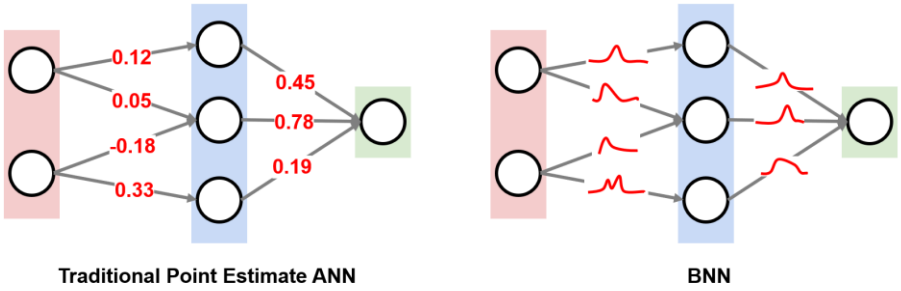

Figure 1. Left: Illustration of traditional point estimate ANN where weights are fixed values. **Right:** Illustration of BNN where weights are assigned distributions.

There are two main advantages of BNNs. Firstly, as the output of BNNs (*i.e.*, the posterior predictive) is a probability distribution, it enables us to quantify the uncertainty of predictions. Secondly, as Bayesian inference takes the prior into account, the impact of the prior on avoiding overfitting is somehow equivalent to those regularization techniques which are often used in training traditional point estimate ANNs. Therefore, BNNs have great potential to address the aforementioned problems in traditional ANNs.

To pursue remarkable performance, ensemble learning strategies are integrated into the training algorithm of ANNs in recent years. Generally speaking, ensemble learning refers to the methodology which combines multiple different machine learning models into an ensemble. By doing so, models in the ensemble are able to do predictions based on the same input. After obtaining a batch of outputs from the member models in the ensemble, the final prediction result of ensemble models can be determined by various approaches such as voting, averaging, or even complex regression analysis. Many analyses have proved that, compared with a single model, an ensemble model has better generalization performance. However, most of the current work focuses on employing ensembling techniques in training traditional ANNs. It is foreseeable that ensemble learning strategies can be used in training BNNs and helping to improve performance as well.

D. Project Objectives

The following sections of this dissertation aim to explore the potential of BNNs trained with various ensemble learning strategies to overcome the inherent defects of traditional point estimate ANNs. Some

previous work related to BNNs and ensembling techniques would be reviewed (section II). Then a variety of training algorithms for BNNs are programmed using Pytorch for implementing experiments (section III). Next, I will evaluate the performance of these algorithms and make a comparison (section IV). Finally, I conclude with a brief discussion based on the results of the experiments (section V).

II. RELATED WORK

In this section, I make a brief literature review on some cutting-edge work related to Bayesian inference algorithms and ensemble learning strategies for ANNs.

A. Bayesian Inference Algorithm

According to the theory of BNNs, the network weights seemingly just need to be sampled from the posterior distribution which is inferred by Bayes' formula without a separate optimization process. However, the computation of the marginal likelihood (evidence) term of Bayes' formula is a difficult problem when using Bayes inference. Due to the complexity of ANNs' architecture and the large number of network weights, the Bayesian posterior distributions are often multimodal and high-dimensional. In practical cases, obtaining the true Bayesian posterior distributions is a computationally expensive even impossible task. Nevertheless, there remain two available methods that can be used to approximate the posterior in machine learning literature. The first is Markov chain Monte Carlo (MCMC) and the second is variational inference (VI). In the literature, the process used to approximate the Bayesian posterior distribution of network weights in BNNs is referred to as training (Wilson and Izmailov, 2020).

1) Markov chain Monte Carlo

Like other Monte Carlo methods, the basic idea behind MCMC is to generate as many samples as possible from the posterior distribution to estimate the posterior. However, instead of inversion or rejection sampling approaches, a Markov chain model is introduced into the Monte Carlo algorithm, which is a sequence of samples, and the state of the i-th sample S_i merely depends on its previous sample S_{i-1}. It can be denoted as:

$$S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \cdots \Rightarrow S_{i-1} \Rightarrow S_i. \tag{6}$$

After an initial burn-in time, the sample of a Markov chain converges to a stationary distribution which is usually very close to the true Bayesian posterior distribution $P(w|D)$. In other words, the samples from this stationary distribution are approximately equivalent to those from the true Bayesian posterior.

However, not all MCMC algorithms are suitable for training BNNs. The most popular of them in BNNs is the Metropolis-Hasting algorithm (this will be further discussed in section III). Furthermore, it should be noted that, as a large number of samples have to be generated and stored before the expected posterior probability distribution is obtained, almost all MCMC approaches have to take quite a long time and many computing resources.

Based on the Metropolis-Hasting algorithm, a novel algorithm called Hamiltonian Monte Carlo (HMC) is developed and applied in BNNs (Neal, 2011). Although HMC inherits most features of the standard Metropolis-Hasting algorithm, the distinctive part of HMC is that it uses a clever sampling scheme, which increases the acceptance rate of samples. Therefore, HMC makes the Markov chain converge faster to the stationary distribution. It significantly improves the operating efficiency of the program. Today, HMC serves as the gold standard inference method for BNNs (Cobb and Jalaian, 2020).

2) Variational Inference

Now that MCMC is a very computationally heavy method to train BNNs, it is necessary to find another method that is able to do Bayesian inference taking little computing resources, namely variational inference. VI refers to the method which gets samples from an intermediate distribution $q(w|\theta)$ rather than the true Bayesian posterior distribution $P(w|D)$ (Hinton and Van Camp, 1993). $q(w|\theta)$ is called the variational posterior and θ represents the parameters of the variational posterior. Therefore, in terms of the BNNs training algorithm using VI method, its main task is to make the variational posterior as close as possible to the true Bayesian posterior through optimizing θ.

Usually, the Kullback-Leibler (KL) divergence is employed to measure the closeness between the

variational posterior and the true Bayesian posterior.

$$\begin{aligned} KL[q(w|\theta) \parallel P(w|D)] &= \int q(w|\theta) log \frac{q(w|\theta)}{P(w|D)} dw \\ &= \mathbb{E}_q \left[log \frac{q(w|\theta)}{P(w)} - logP(D|w) \right] + logP(D) \\ &= KL[q(w|\theta) \parallel P(w)] - \mathbb{E}_q[logP(D|w)] + logP(D). \end{aligned} \tag{7}$$

To find the optimal parameters θ^*, we need to minimize the KL divergence between $q(w|\theta)$ and $P(w|D)$. Moreover, $logP(D)$ in formula (7) is a constant, so θ_* can be computed by:

$$\theta^* = arg\ min_\theta\ KL[q(w|\theta) \parallel P(w)] - \mathbb{E}_q[logP(D|w)]. \tag{8}$$

The formula (8) enables us to construct a loss function in the VI algorithm, which is defined as:

$$\mathcal{F}(D, \theta) = KL[q(w|\theta) \parallel P(w)] - \mathbb{E}_q[logP(D|w)]. \tag{9}$$

$\mathcal{F}(D, \theta)$ has various names in literature. It can be called the variational free energy (Friston *et al.*, 2007) or the expected lower bound (Jaakkola and Jordan, 2000). And $-\mathcal{F}(D, \theta)$ can also be used as an objective function which is referred to as the evidence lower bound (ELBO). After all, minimizing $\mathcal{F}(D, \theta)$ is equivalent to maximizing $-\mathcal{F}(D, \theta)$. As an aside, $KL[q(w|\theta) \parallel P(w)]$ is often called the complexity loss which is associated with the prior, and $-\mathbb{E}_q[logP(D|w)]$ is called the likelihood loss associated with data. However, as $\mathcal{F}(D, \theta)$ is computationally prohibitive, Blundell *et al.* (2015) put forward a method to approximate it.

$$\mathcal{F}(D, \theta) \approx \sum_{i=1}^{n} log\, q\left(w^{(i)}|\theta\right) - logP\left(w^{(i)}\right) - logP\left(D|w^{(i)}\right). \tag{10}$$

With this loss function (10), VI can be adapted to the classical backpropagation training algorithm of traditional ANNs. This new backpropagation-compatible algorithm is called Bayes by Backprop (Blundell *et al.*, 2015), which is the VI algorithm that I am going to use in the next section.

B. Ensemble Learning Strategies

Traditionally, people like to combine models having different architectures into an ensemble and let them do predictions based on the same input. This is because the large diversity of models makes the whole ensemble consider as many potential results as possible, which is helpful to improve generalization performance. The idea of ensemble learning exactly makes sense and takes effect in ANNs. However, the price of designing and training totally different models is expensive. Some recent work proves that it will be feasible as well if we combine models having the same architecture but different parameters.

1) Snapshot Ensembling

Huang *et al.* (2017) develop a novel ensembling strategy called Snapshot Ensembling. They take snapshots for the network weights at different times when they train an ANN model using a cyclical cosine annealing learning rate schedule. After training ends, an ensemble model is created by combining multiple single models. These member models have the same network architecture, but they are loaded with sets of weights that are stored at different times. Compared with those ensembling strategies which train member models having totally different architectures, Snapshot Ensembling is a relatively cheap way because it actually trains only one ANN model and records network weights from time to time during the training process.

In most cases, a snapshot ensemble model has better test performance than that of a single model. As Figure 1 shows, the loss function of ANNs is usually non-convex. For a given single model trained using SGD, it tends to be trapped in local optima at the end of training (see the left side of Figure 2), and it is rarely able to find the global optima. If we train the ANN model using a cyclical learning rate (see the right side of Figure 2), some different solutions of local optima are easy to obtain. In Snapshot Ensembling, every point of local

optima represents a high-performing model, then these points of local optima comprise an ensemble model which often has better generalization performance when compared with a single model. Generally speaking, the interval between two adjacent snapshots is equal to the cycle length of the learning rate schedule (around 40 to 60 epochs per cycle). This is to guarantee that sufficiently different sets of weights can be collected. Therefore, the total number of epochs required for Snapshot Ensembling could be large.

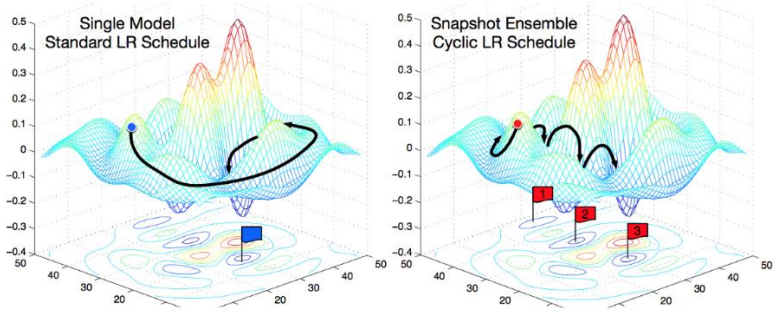

Figure 2. Train loss surface of an ANN as a function of network weights. **Left:** SGD optimization with a typical learning rate schedule. **Right:** Snapshot Ensembling. (Huang *et al.*, 2017)

2) Fast Geometric Ensembling

Based on the theory of Snapshot Ensembling, Garipov *et al.* (2018) put forward an improved strategy called Fast Geometric Ensembling (FGE). Instead of the cyclical cosine annealing learning rate schedule applied in Snapshot Ensembling, FGE employs a linear piecewise cyclical learning rate. FGE also collects member models through recording weights from time to time when training an ANN model. And the interval depends on the cycle length of the learning rate schedule as well. The distinguishing part is that the cycle length of the learning rate schedule in FGE is very short (usually below 10 epochs per cycle). Intuitively, short cycle length could inevitably lead to the high similarity among member models. However, Garipov *et al.* claim that, for a successively trained ANN, points of local optima on the loss surface are not separated by the high loss region as shown on the left of Figure 3. The truth is that, on the high-dimensional loss surface, there exists a low loss path that connects the different points of local optima (see the middle and right of Figure 3). Therefore, if the model travels along this low loss path in small steps, it will be very likely to collect many sufficiently different high-performing models to comprise an ensemble model which can provide a great prediction result.

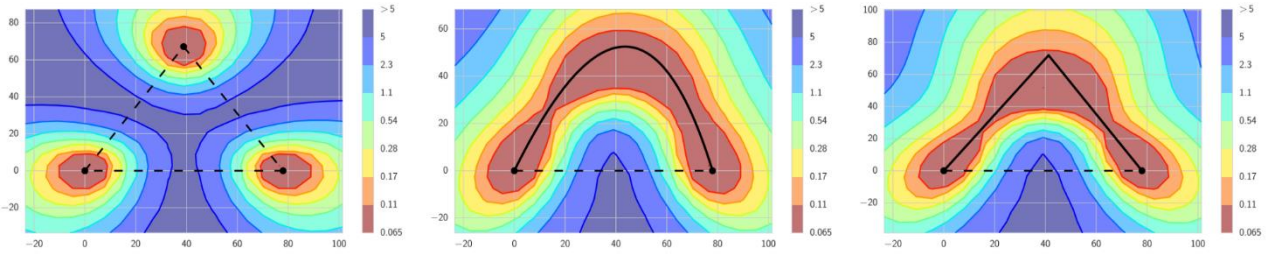

Figure 3. Train loss surface of an ANN as a function of network weights. **Left:** Three optima for independently trained networks. **Middle and Right:** Two paths of low loss, connecting the lower two optima on the left panel. (Garipov *et al.*, 2018)

3) Stochastic Weight Averaging

Unlike the model averaging approach used in all aforementioned ensemble learning strategies, a novel weight averaging approach called Stochastic Weight Averaging (SWA) is created by Izmailov *et al.* (2018). What SWA does is just averaging the weights which are collected at different stages when training the same ANN model with a high constant or cyclical learning rate schedule. When using an SWA model for prediction, the model would be simply loaded with averaged weights. Hence, SWA doesn't take any additional cost over traditional training. Although the model trained using SWA is, in fact, a single model rather than an ensemble model, the idea of SWA follows the trajectory of ensemble learning strategies, so SWA can also be considered as one of them.

As Figure 4 shows, the point of optima found by SGD usually is on the boundary of the wide flat region of low loss, and the central point of this flat region can be obtained by averaging the points near the boundary.

Hence, the optima found by SWA is wider than that of SGD. Generally, the width of the optima is related to generalization performance (Keskar *et al.*, 2016). Overall, SWA can provide a solution of better generalization performance, and SWA costs nearly no computational overhead compared to traditional training schemes.

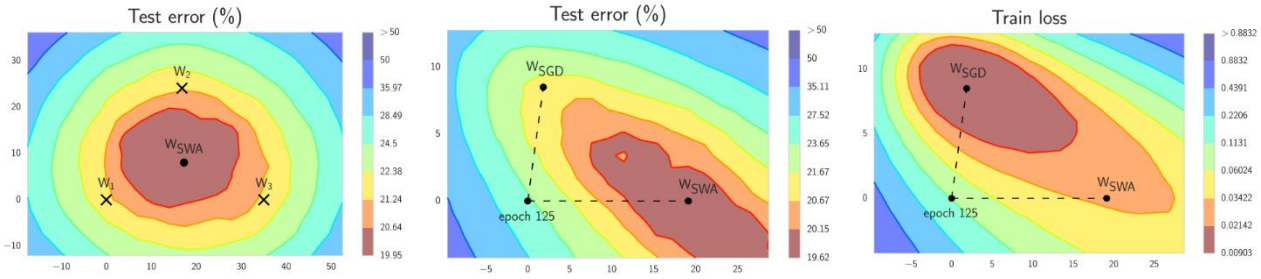

Figure 4. Left: W_1, W_2, W_3 represent 3 SGD solutions on the test error surface of an ANN, and W_{swa} represents the corresponding SWA solution. **Middle and Right:** The SGD and SWA training trajectories of an ANN on test error and train loss surfaces, starting from the same initialization of SGD after 125 training epochs. (Izmailov *et al.*, 2019)

III. SOLUTION

This section is the solution to determine whether BNNs are capable of avoiding overfitting and providing the uncertainty associated with predictions and to explore the influence of various ensemble learning strategies on improving the performance of BNNs. In this section, I present two works. In the first work, a simple multi-layer perceptron (MLP) network is designed, which has only one hidden layer. Then this MLP network is trained on the MNIST dataset using different Bayesian inference approaches so that I can obtain different types of BNN models. The performance comparison is made among the resulting BNN models. In the second work, a convolutional neural network (CNN) is designed, then this CNN is trained using the VI approach (Bayes by Backprop). Additionally, a variety of ensembling strategies are applied in the algorithm. And the performance of different ensemble models is also compared. All algorithms in this section are coded using Pytorch and executed in Google Colab.

A. Training a Multi-linear Perceptron Network Using Various Methods

1) The architecture of the MLP network

To this day, MNIST has become a Hello World dataset in the literature of ANNs, which consists of 70,000 hand-written digit images of size 28 × 28 pixels in total. Each image is labelled with a specific number between 0 to 9. Generally, 60,000 of them are used as the training set and the rest of them are used as the test set, and I also use this splitting method. The following experiments in this work are based on the MNIST dataset.

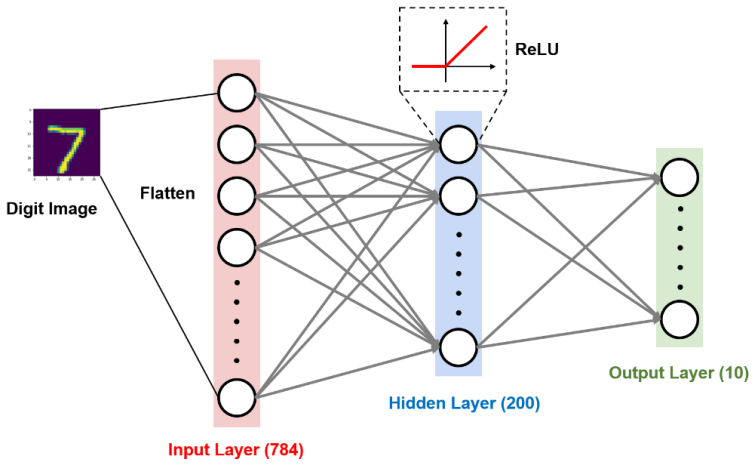

Figure 5. Illustration of the architecture of the MLP network.

In this work, I design an MLP network that acts as an image classifier. The architecture of the MLP network is shown in Figure 5. It is composed solely of three fully connected layers. There are 784 nodes (corresponding to 28 × 28 pixels) in the input layer, 200 nodes in the hidden layer, and 10 nodes (corresponding to 10 classes) in the output layer. ReLU is chosen to be the activation function of the hidden layer. This architecture is relatively simple because I just intend to use it to compare different training methods in this work. If the architecture of the network was too deep, it would cost too many computing resources and too much training time, particularly for MCMC.

2) MCMC (HMC)

The first experiment is training the MLP network using MCMC, which is one of the Bayesian inference methods mentioned in section II. Among all MCMC algorithms, the Metropolis-Hasting algorithm is the most commonly used for BNNs, because it merely requires a function $f(x)$ which is proportional to the posterior $P(w|D)$. The Metropolis-Hasting algorithm begins with an initial guess w_0 which is sampled from an initial proposal distribution. Then a new candidate sample w' is drawn from the current proposal distribution $Q(w'|w_i)$. In most cases, w' is close to the previous sample w_i. This is to increase the acceptance rate of samples. If w' is more likely than w_i according to the true posterior distribution $P(w|D)$, w' will be accepted within a certain rate, otherwise, it will be rejected. The acceptance rate is:

$$P = min\left(1, \frac{f(w')}{f(w_i)}\right). \tag{11}$$

This is the acceptance rate formula of the Metropolis algorithm but it is only fit for the situation where the proposal distribution Q is symmetric, such as a Gaussian distribution. If the Hasting term is introduced into the above equation, it will be transformed to:

$$P = min\left(1, \frac{Q(w'|w_i)}{Q(w_i|w')} * \frac{f(w')}{f(w_i)}\right). \tag{12}$$

This is the acceptance rate formula of the Metropolis-Hasting algorithm which is still available under situations where the proposal distribution is asymmetric. If a new candidate sample is accepted, the proposal probability distribution $Q(w'|w_i)$ will be updated, then the algorithm will repeat the above operations until having collected a sufficient number of samples. The general procedure of the Metropolis-Hasting algorithm is summarized in Algorithm 1.

Algorithm 1. Pseudocode of the Metropolis-Hasting algorithm.

Draw w_0 from the initial proposal probability distribution $Q(w_0)$;
$i \leftarrow 0$;
while $i < N$ **do** # {N is a hyper-parameter which represents the required total number of samples}
  Draw w' from the current proposal probability distribution $Q(w'|w_i)$;
  $P \leftarrow min\left(1, \frac{Q(w'|w_i)}{Q(w_i|w')} * \frac{f(w')}{f(w_i)}\right)$; #{$P$ is the acceptance rate}
  Draw k from $Bernoulli(P)$;
  if k **then**
    $w_{i+1} \leftarrow w'$;
  else
    $w_{i+1} \leftarrow w_i$;
  $i \leftarrow i + 1$;
end while

The coding work of this experiment is implemented with the help of the package hamiltorch which is provided by Cobb and Jalaian (2020). It offers a ready-made framework of HMC having a highly efficient sampling scheme. As mentioned in section II, HMC is based on the Metropolis-Hasting algorithm, so it basically follows the general procedure of the standard Metropolis-Hasting algorithm described in Algorithm 1. After training the MLP network using HMC (1,000 samples in total for each weight), a BNN model is obtained for evaluation.

3) VI (Bayes by Backprop)

The second experiment is training the MLP network using VI. As introduced in section II, there is a variational posterior $q(w|\theta)$ in VI. The true posterior $P(w|D)$ can be approximated by minimizing $KL[q(w|\theta) \parallel P(w|D)]$, which is equivalent to minimizing $\mathcal{F}(D, \theta)$ in formula (10). Therefore, the task of all VI algorithms is to optimize the parameters of the variational posterior, θ, to make it closer to the true Bayesian posterior, and $\mathcal{F}(D, \theta)$, meanwhile, works as the loss function. It motivates the use of gradient-based optimization algorithms (*e.g.*, backpropagation) which usually are used in training traditional ANNs. However, there is an issue that the stochasticity of the sampling process stops backpropagation.

In this experiment, Bayes by Backprop is employed to train the MLP network. There is a reparameterization trick to ensure backpropagation works as usual. For example, supposing the variational posterior is a Gaussian distribution, there would be two parameters which are a mean μ and a standard deviation σ, respectively.

$$w \sim \mathcal{N}(\mu, \sigma). \tag{13}$$
$$\theta = (\mu, \sigma). \tag{14}$$

In Bayes by Backprop, the standard deviation σ is reparameterized as:

$$\sigma = log(1 + e^{\rho}). \tag{15}$$
$$\theta = (\mu, \rho). \tag{16}$$

Next, we need a random variable ϵ as the source of noise, which is sampled from a diagonal Gaussian distribution $\mathcal{N}(0, I)$. Thus, a posterior sample of weights w can be generated by a deterministic function $t(\theta, \epsilon)$.

$$\epsilon \sim \mathcal{N}(0, I). \tag{17}$$
$$w = t(\theta, \epsilon) = \mu + log(1 + e^{\rho}) \odot \epsilon. \tag{18}$$

By doing so, classical backpropagation algorithm can be compatible with VI. And the parameters of the variational posterior can be optimized by Stochastic Gradient Descent (SGD). The general procedure of Bayes by Backprop is summarized in Algorithm 2.

Algorithm 2. Pseudocode of Bayes by Backprop.

$\theta \leftarrow \theta_0$;
for $i \leftarrow 1, 2, \ldots, N$ **do** #{N is the number of epochs}
  Draw ϵ from $\mathcal{N}(0, I)$;
  $t(\theta, \epsilon) \leftarrow \mu + log(1 + e^{\rho}) \odot \epsilon$;
  $w \leftarrow t(\theta, \epsilon)$;
  $f(w, \theta) \leftarrow logq(w|\theta) - logP(w) - logP(D|w)$; #{Loss function}
  Backpropagation:
  $\triangle_{\mu} \leftarrow \frac{\partial f(w,\theta)}{\partial w} + \frac{\partial f(w,\theta)}{\partial \mu}$;
  $\triangle_{\rho} \leftarrow \frac{\partial f(w,\theta)}{\partial w} \frac{\epsilon}{1+e^{-\rho}} + \frac{\partial f(w,\theta)}{\partial \rho}$;
  SGD:
  $\mu \leftarrow \mu - \alpha \triangle_{\mu}$;
  $\rho \leftarrow \rho - \alpha \triangle_{\rho}$;
end for

By the way, a Gaussian variational posterior and a scale mixture prior are used in the algorithm of this experiment, and the coding work is implemented according to the procedures shown in Algorithm 2. After 10 epochs of training by Bayes by Backprop with a constant learning rate (4e-3), a BNN model is obtained for evaluation.

4) Other Methods

Apart from the two Bayesian inference method experiments described above, there still are 3 other experiments implemented in this work in order to make a comparison with them:

(1) **Classical Backpropagation:** The MLP network is trained in the traditional way (*i.e.*, backpropagation and SGD). The output model is a point estimate ANN model used as a control in this work.

(2) **Classical Ensemble:** The MLP network is trained in the traditional way as same as (1), but the training program is executed 10 times in this experiment. Thus, 10 independently trained point estimate models are obtained and then gather into an ensemble model for evaluation.

(3) **VI Ensemble:** The MLP network is trained using Bayes by Backprop, and the training program is also executed 10 times in this experiment. Eventually, 10 independently trained BNN models are obtained and then gather into an ensemble model for evaluation.

B. Training a Bayesian Convolutional Neural Network Using Various Ensemble Learning Strategies

1) The Architecture of BayesianLeNet-5

In this work, a convolutional neural network (CNN) is trained using Bayes by Backprop, so the model it outputs would be a Bayesian convolutional neural network (BCNN) model. As Figure 6 shows, the architecture of this BCNN is composed of 2 convolutional layers and 3 linear fully connected layers, and each convolutional layer is followed by a max-pooling layer. Although this architecture is not exactly the same as the classical CNN LeNet-5 invented by LeCun *et al.* (1998), this BCNN model is named BayesianLeNet-5. Then different ensemble learning strategies are applied in the training algorithm of BayesianLeNet-5, and the performance of ensemble models they output is evaluated and compared.

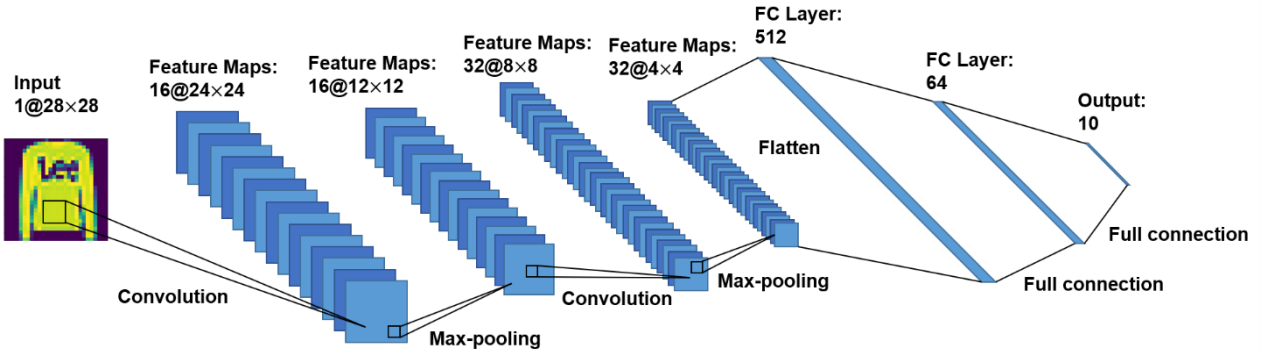

Figure 6. Illustration of the architecture of BayesianLeNet-5.

In addition, the dataset on which BayesianLeNet-5 is trained in this work is the Fashion-MNIST dataset. Fashion-MNIST consists of 70,000 grayscale images of size 28 × 28 pixels, and each image is a picture of one of the items of clothing. There are 10 classes of clothing in Fashion-MNIST, such as T-shirt, trouser, pullover, etc. And each class of clothing is labelled by an integer from 0 to 9. Although Fashion-MNIST shares the same size and format as MNIST, Fashion-MNIST is a more challenging dataset for an image classification task. Therefore, Fashion-MNIST is a perfect replacement for MNIST for benchmarking a variety of ensemble learning algorithms in this work.

Furthermore, in this work, 50,000 images of Fashion-MNIST are used as the training set, 10,000 images are used as the validation set, and the remaining 10,000 images are used as the test set.

2) Traditional Ensembling

In the first experiment of this work, the traditional ensembling strategy is applied in the BayesianLeNet-5 training algorithm. It means that BayesianLeNet-5 is trained 30 epochs using Bayes by Backprop with a monotone cosine annealing learning rate schedule (see Figure 7), and the training program is executed 10 times in total. Thus, there will be 10 independently trained BayesianLeNet-5 models gathered into an ensemble

model for evaluation. Although it is mentioned in section II that traditional ensembles usually like to collect models which have different architectures, the member models of the traditional ensemble in this experiment are all based on the architecture of BayesianLeNet-5 but they have different network weights. This is to control variables so that it can be well compared with other ensemble learning strategies.

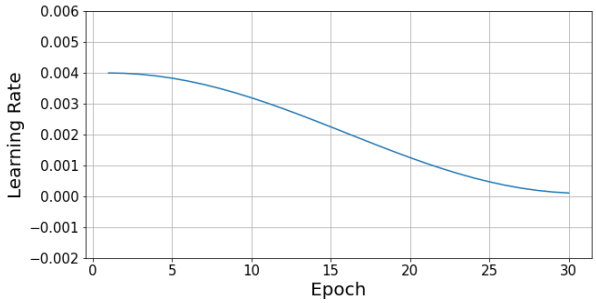

Figure 7. The learning rate schedule of each independently trained model in the traditional ensemble.

3) Snapshot Ensembling

In the second experiment of this work, Snapshot Ensembling is applied in the BayesianLeNet-5 training algorithm. As Figure 8 shows, BayesianLeNet-5 is successively trained 300 epochs using Bayes by Backprop with a cyclical cosine annealing learning rate schedule, and the cosine annealing cycle length of learning rate is constantly 30 epochs. Furthermore, the network weights are termly saved at the time when the learning rate decays to the minimum value (1e-4) because the model generally performs well at that time. After training is finished, there are 10 sets of network weights are saved in total. Then an ensemble model can be obtained for evaluation by loading these saved network weights into 10 BayesianLeNet-5 member models separately.

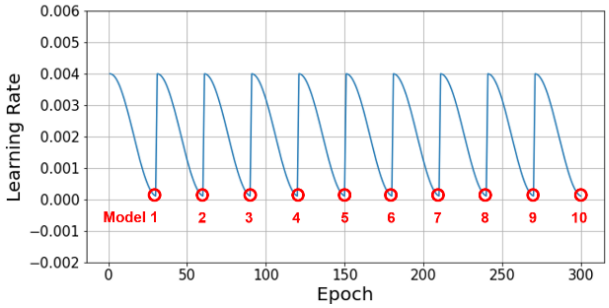

Figure 8. The learning rate schedule of Snapshot Ensembling; red circles represent the time when network weights are saved.

4) FGE

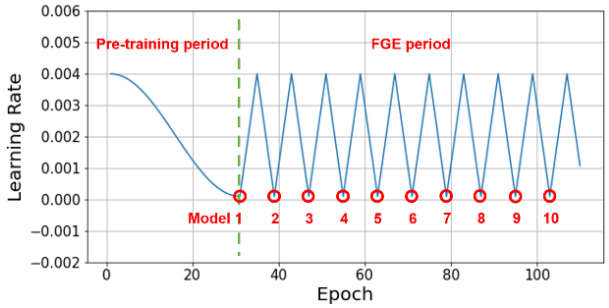

Figure 9. The learning rate schedule of FGE; red circles represent the time when network weights are saved.

In the third experiment of this work, FGE is applied in the BayesianLeNet-5 training algorithm. The training can be divided into two periods, namely the pre-training period and the FGE period (see Figure 9). During the pre-training period (1-30 epoch), BayesianLeNet-5 is trained using Bayes by Backprop with a monotone cosine annealing learning rate schedule. Afterwards, a linear piecewise cyclical learning rate schedule is used in the FGE period (31-110 epoch), and the cycle length is 8 epochs. Then the network weights are termly saved at the time when the learning rate decays to the minimum value (1e-4). After training is

finished, 10 sets of network weights are loaded into 10 BayesianLeNet-5 models separately, which comprise an ensemble model for evaluation. Therefore, the general procedure of FGE is quite similar to that of Snapshot Ensembling, but the cycle length in FGE is shorter than that in Snapshot Ensembling.

5) SWA

In the last experiment of this work, SWA is applied in the BayesianLeNet-5 training algorithm. Similarly, SWA training can also be divided into two periods (see Figure 10), which are the pre-training period and the SWA period, respectively. In the pre-training period (1-15 epoch), BayesianLeNet-5 is trained using Bayes by Backprop with a monotone cosine annealing learning rate schedule. After 15 epochs, the learning rate gradually increases to a high constant value (6e-3) in 5 epochs. Then the learning rate keeps at this constant value until training is finished. In the SWA period (16-30 epoch), the network weights at each epoch are all saved. Moreover, these saved network weights are averaged pointwise, then the averaged weights are loaded into an SWA model of BayesianLeNet-5. And this SWA model is used for prediction and evaluation. Therefore, SWA is essentially an ensembling method in weight space.

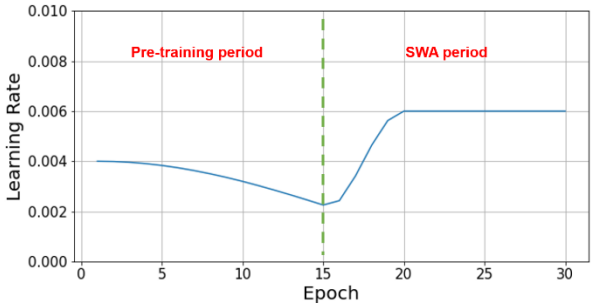

Figure 10. The learning rate schedule of SWA.

IV. Results and Evaluation

In this section, the results of all experiments are presented and evaluated. There are two subsections. The first subsection focuses on the comparison among different training methods. And the second subsection concentrates on comparing different ensemble learning strategies.

A. Training Methods Comparison

This subsection aims to study whether BNNs can provide great prediction performance and uncertainty estimates. The MLP network model is trained using different methods on the MNIST dataset in all experiments of this subsection, and the results are shown in the following part.

1) MCMC (HMC)

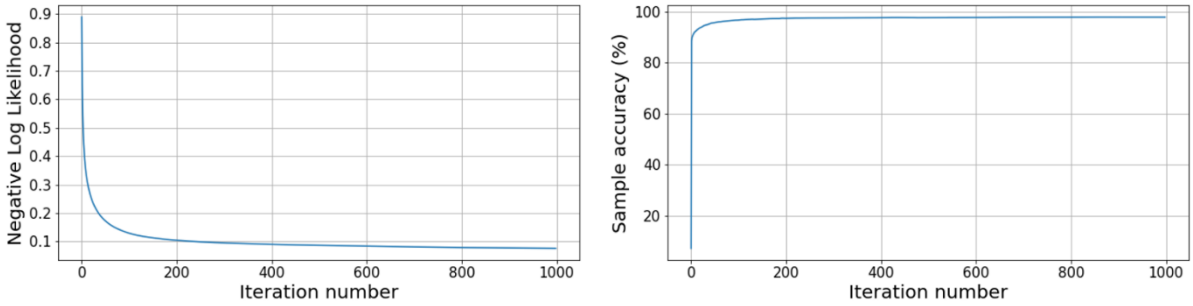

Figure 11. Left: The negative log-likelihood of the MLP model trained by HMC as a function of iteration number. **Right:** The prediction accuracy of the MLP model trained by HMC as a function of iteration number.

As Figure 11 presents, when using HMC, there are a total of 1,000 accepted samples drawn from the Bayesian posterior and the mean acceptance rate is 96%. It demonstrates that HMC exactly has a sampling approach of high efficiency compared to other MCMC algorithms in the literature. As the number of accepted

samples (iteration number) increases, the negative log-likelihood (NLL) of the MLP model gradually drops and the prediction accuracy on the test set rises. Both of them basically converged after 200 samples are collected. Eventually, in terms of the BNN model trained using HMC, its prediction accuracy on the test set is 97.84%. It indicates that HMC is a very great approach to training a BNN model of high performance.

2) VI (Bayes by Backprop)

As Figure 12 shows, when using Bayes by Backprop to train the MLP model, although the train loss (*i.e.*, $\mathcal{F}(D,\theta)$ or $-ELBO$, see the formula (10)) is still decreasing, the prediction accuracy on the test set has fast converged after the first epoch. After all, MNIST is not a very challenging imaging classification task. The prediction accuracy of the output BNN model on the test set is 97.49%.

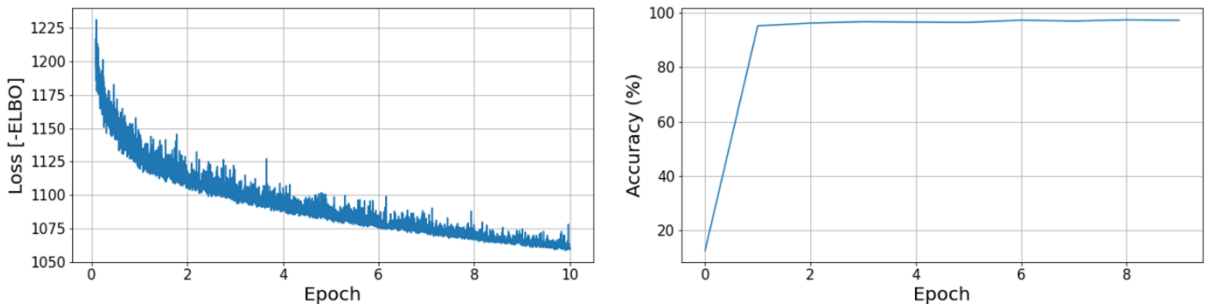

Figure 12. Left: The train loss of MLP model trained by Bayes by Backprop as a function of training epoch. **Right:** The prediction accuracy of MLP model trained by Bayes by Backprop as a function of training epoch.

In order to further explore the behaviour of the loss curve, each loss component is measured and presented in Figure 13 separately. As introduced in the section II, the loss function of (10) can be explained as a sum of a prior-dependent component $logq(w|\theta) - logP(w)$, which is called the complexity loss, and a data-dependent component $-logP(D|w)$, which is referred to as the likelihood loss. The loss function $\mathcal{F}(D,\theta)$ reflects the balance between making the variational posterior $logq(w|\theta)$ close to the prior $P(w)$ and satisfying the complexity of the data D. The results in Figure 13 show that the complexity loss slowly decreases, while the likelihood loss (*i.e.*, the negative log-likelihood) converges very fast. It indicates that in the Bayes by Backprop algorithm, the network weights are priorly optimized for the data part before shifting to finding a great distribution around the means.

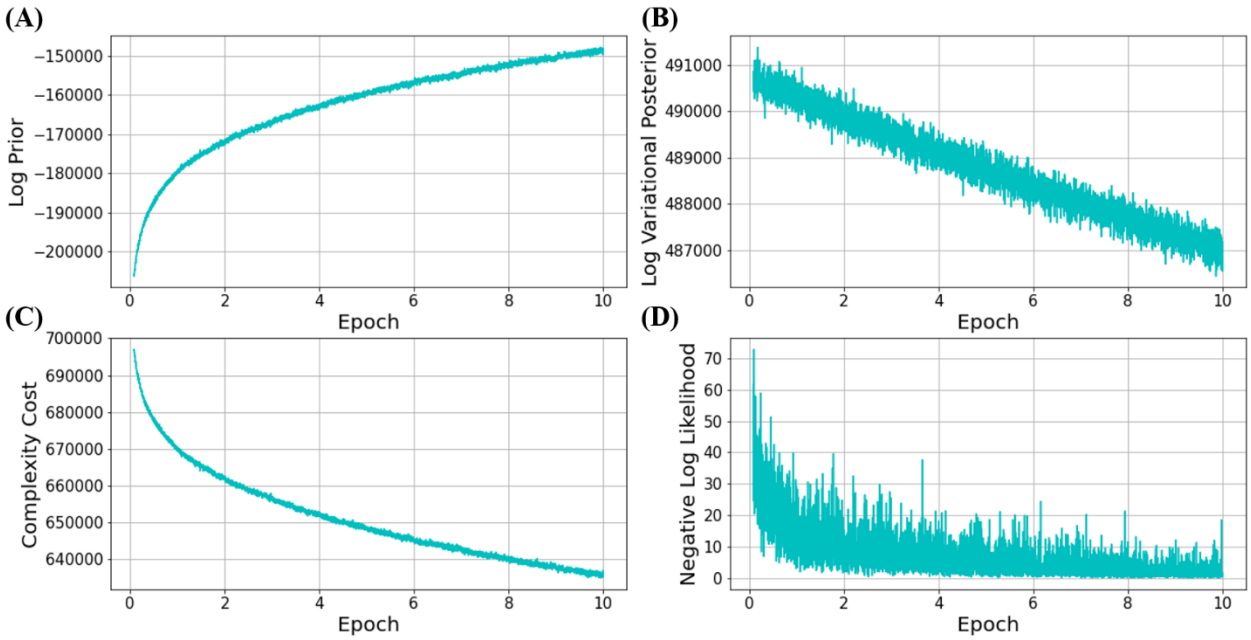

Figure 13. (A): The log prior of the MLP model as a function of training epoch for Bayes by Backprop. **(B):** The log variational posterior of the MLP model as a function of training epoch for Bayes by Backprop. **(C):** The complexity loss of the MLP model as a function of training epoch for Bayes by Backprop. **(D):** The negative log-likelihood of the MLP model as a function of training epoch for Bayes by Backprop.

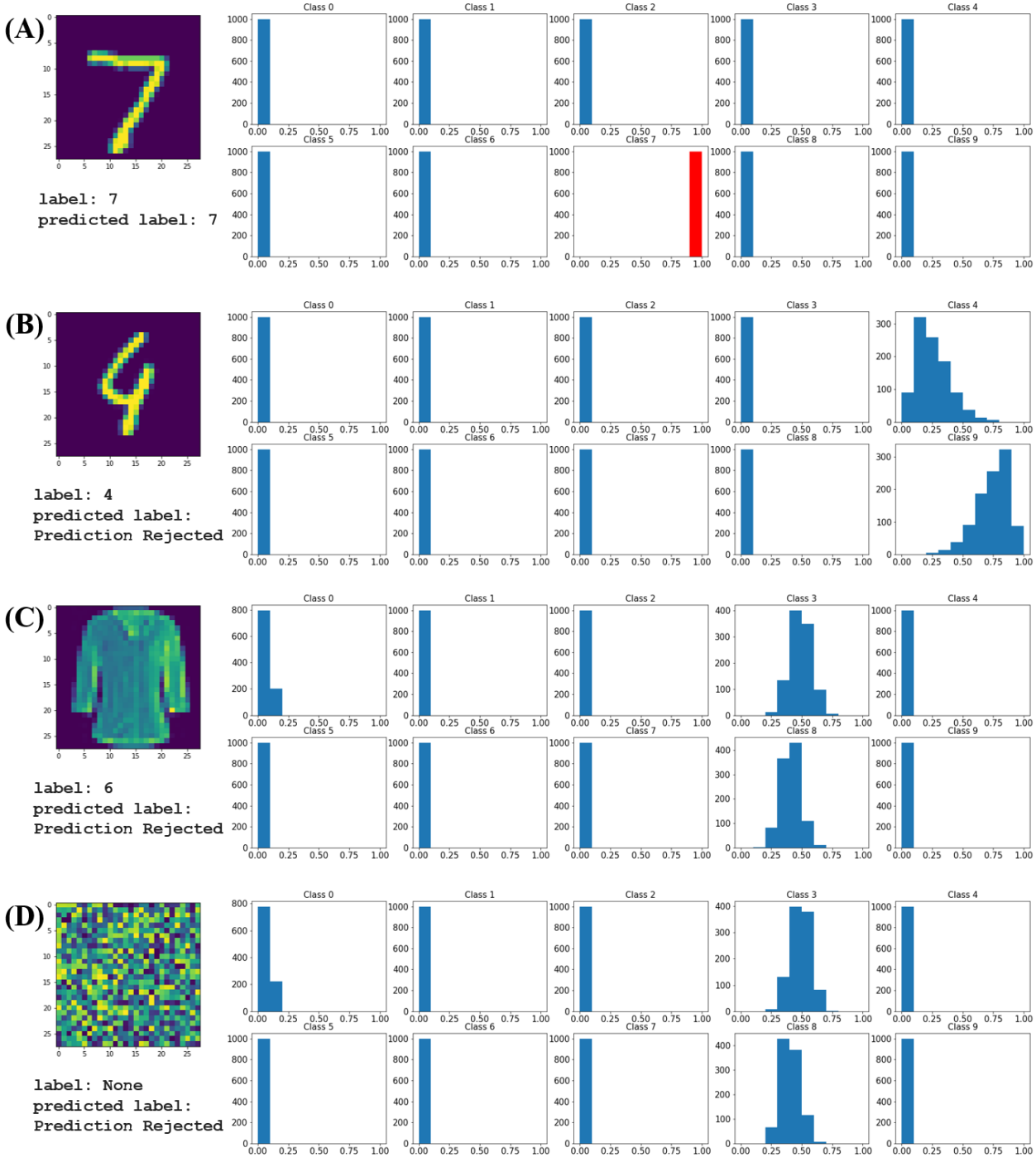

Figure 14. **(A):** Histograms of prediction of Bayesian MLP based on an MNIST image input (correctly labelled). **(B):** Histograms of prediction of Bayesian MLP based on an MNIST image input (incorrectly labelled). **(C):** Histograms of prediction of Bayesian MLP based on a Fashion-MNIST image input. **(D):** Histograms of prediction of Bayesian MLP based on a white noise image input.

In addition, one of the major concerns is whether BNN models are capable of quantifying the uncertainty associated with predictions. Hence, after being trained using Bayes by Backprop, the model (called Bayesian MLP) is employed to do image classification work based on a variety of image inputs. In this experiment, I use the Bayesian MLP model to predict the label of each image 1,000 times. Then the model randomly sample network weights from the variational Bayesian posterior distribution at each time. Moreover, the final prediction results will go through a softmax function before they are outputted. Thus, based on an input image, there are 1,000 prediction results for each class. Finally, for four types of input images, the histograms of prediction results for each class are shown in Figure 14.

For an MNIST image that is correctly labelled by the model (see Figure 14(A)), the Bayesian MLP model has sufficient confidence in its prediction because the histograms indicate that all 1,000 predictions classify this image as digit 7 (marked in the red bar). Furthermore, for an MNIST image incorrectly labelled by the model in Figure 14(B), the Bayesian MLP shows its uncertainty through the histograms, which indicate that the model vacillates between digit 4 and digit 9. The hesitation of the model coincides with the feeling of a human observer for this image. The model finally rejects the request for prediction on this image, because I set a rule in the prediction algorithm. The rule is that the model will refuse to predict the image label if the averaged output values for each class are all below 0.95. Similarly, when facing images that are out of the dataset that the model was trained on, such as a Fashion-MNIST image in Figure 14(C) and a white noise image in Figure 14(D), the model shows its uncertainty in the prediction histograms and refuse the request for prediction. Overall, Bayesian MLP has the ability to quantify the uncertainty of predictions.

3) Classical Ensemble

As Figure 15 shows, there are 10 MLP network models independently trained using classical backpropagation. Even though these models have the same architecture, there still are some differences among them during the training process because they have different initial network weights. And these 10 point estimate ANN models gather into an ensemble model after training is finished. The prediction accuracy of this ensemble model on the test set is 96.70%.

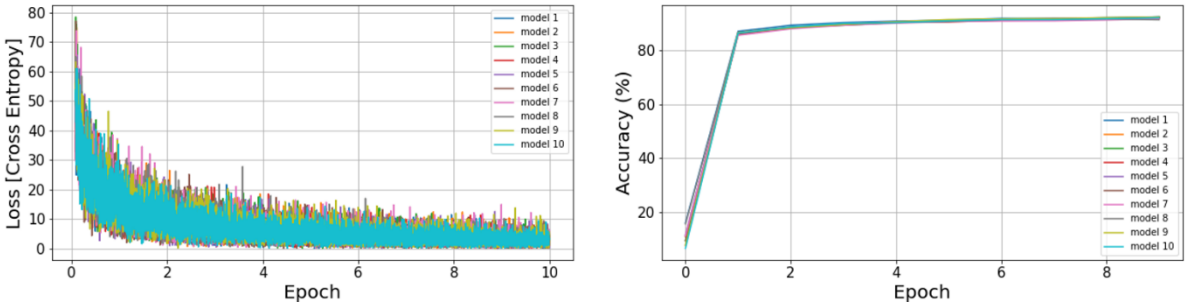

Figure 15. Left: The train loss of the MLP models independently trained by classical backpropagation as a function of training epoch. **Right:** The prediction accuracy of the MLP models independently trained by classical backpropagation as a function of training epoch.

4) VI Ensemble

Similarly, for 10 MLP network models with different initial network weights, there also are some differences among them during the training process, when being trained using Bayes by Backprop (see Figure 16). Then these 10 BNN models are collected into an ensemble model. The prediction accuracy of this ensemble model on the test set is 98.38%.

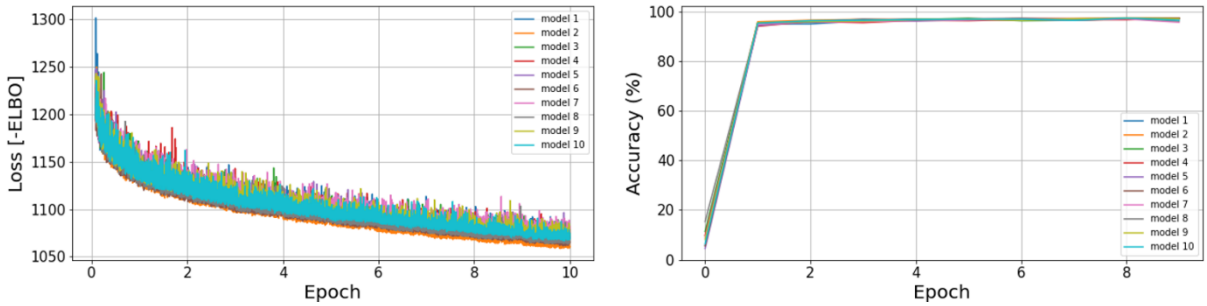

Figure 16. Left: The train loss of the MLP models independently trained by Bayes by Backprop as a function of training epoch. **Right:** The prediction accuracy of the MLP models independently trained by Bayes by Backprop as a function of training epoch.

5) Summary

Eventually, all training algorithms are benchmarked and the performance of their output models is evaluated, and the results are presented in Table 1 and Figure 17. There are some conclusions that can be summarised from it:

(1) Compared with the point estimate ANN model trained using classical backpropagation, BNN models trained using HMC or Bayes by Backprop have better generalization performance.

(2) HMC can provide the BNN model with better generalization performance compared to Bayes by Backprop. This is because HMC directly gets samples from the true Bayesian posterior, but Bayes by Backprop gets samples from the variational posterior which is just an approximation.

(3) However, HMC requires much more training time and its training process is the slowest among all algorithms.

(4) Ensemble models have better generalization performance than those of single member models but training an ensemble model takes more time because it has to train multiple member models.

Table 1. Comparison of training time and prediction accuracy among different training methods.

Training Method	Training Time (s)	Prediction Accuracy on the Test Set (%)
Classical Backpropagation (control)	162	90.64
MCMC (HMC)	17306	97.84
VI (Bayes by Backprop)	476	97.49
Classical Ensemble	1666	96.70
VI Ensemble	4850	98.51

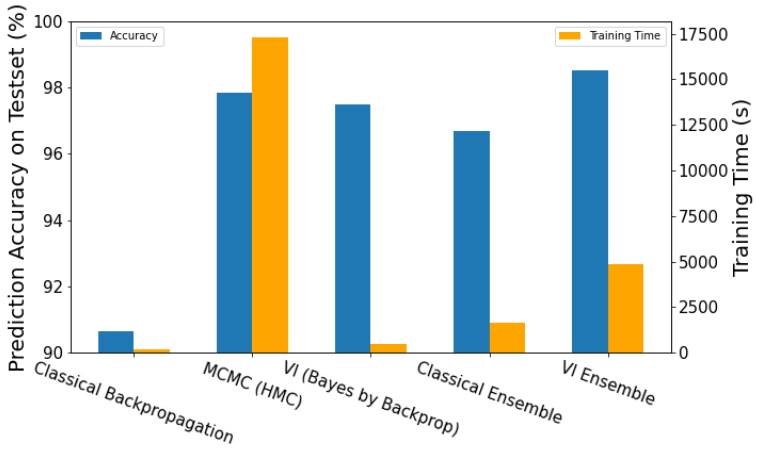

Figure 17. The combined bar chart of prediction accuracy of models trained using different methods on the test set (blue bar) and training time for different training algorithms (orange bar).

B. Ensemble Learning Strategies Comparison

This subsection aims to explore the influence of different ensemble learning strategies on the performance of BNNs. The BayesianLeNet-5 model is trained using Bayes by Backprop with different ensemble learning strategies on the Fashion-MNIST dataset in all experiments of this subsection, and the results are shown in the following part.

1) Traditional Ensembling

In the training algorithm with traditional ensembling, 10 BayesianLeNet-5 models are independently trained using Bayes by Backprop. As Figure 18 shows, the train loss curve and prediction accuracy curve of every single model are slightly different from each other due to different initial network weights. And the prediction accuracy curves of all single models on the validation set basically start to converge after the 5th epoch. Then these member models comprise an ensemble model. The prediction accuracy of this ensemble model on the test set is 91.48%.

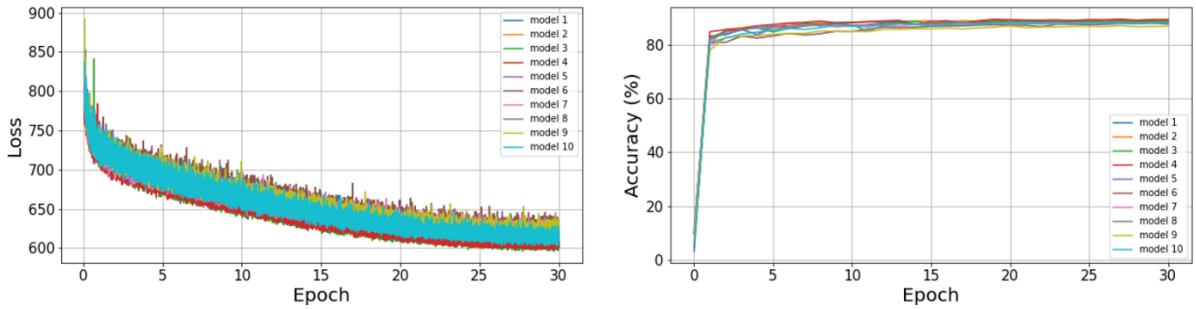

Figure 18. Left: The train loss of BayesianLeNet-5 models independently trained by Bayes by Backprop as a function of training epoch. **Right:** The prediction accuracy of BayesianLeNet-5 models independently trained by Bayes by Backprop as a function of training epoch.

2) Snapshot Ensembling

As a result of a cyclical cosine annealing learning rate schedule used in the training algorithm, the train loss curve and prediction accuracy curve of the BayesianLeNet-5 model for Snapshot Ensembling also show a cyclical feature. As Figure 19 shows, when the learning rate decays to the minimum value, the train loss reaches the minimum value in a cycle and the prediction accuracy on the validation set rises to the maximum value in a cycle. Hence, these points correspond to high-performing networks which are saved as member models of an ensemble. Eventually, the prediction accuracy of this ensemble model on the test set is 88.91%.

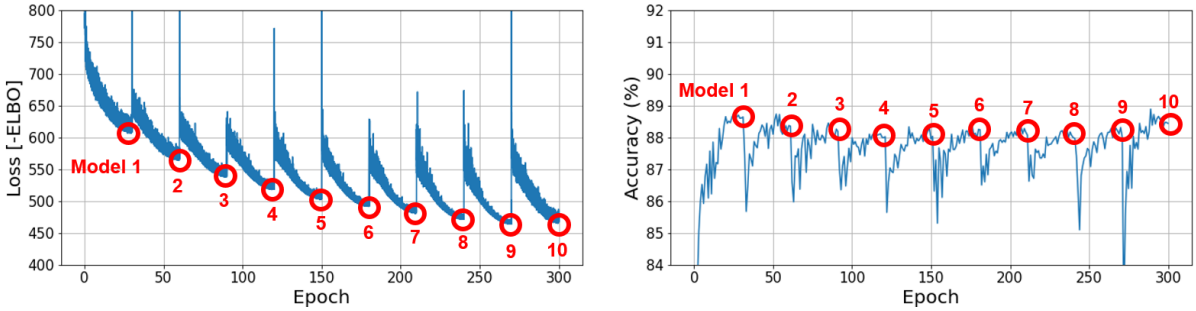

Figure 19. Left: The train loss of BayesianLeNet-5 model trained by Bayes by Backprop with Snapshot Ensembling as a function of training epoch. **Right:** The prediction accuracy of BayesianLeNet-5 model trained by Bayes by Backprop with Snapshot Ensembling as a function of training epoch. Red circles represent the time when network weights are saved.

3) FGE

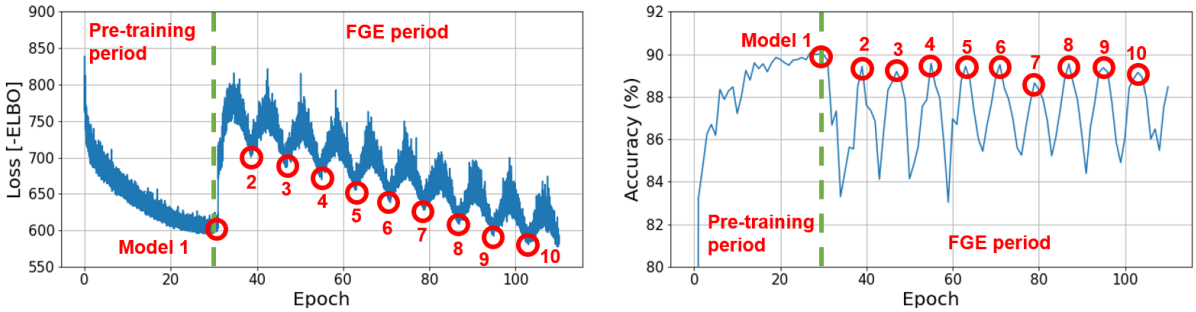

Figure 20. Left: The train loss of BayesianLeNet-5 model trained by Bayes by Backprop with FEG as a function of training epoch. **Right:** The prediction accuracy of BayesianLeNet-5 model trained by Bayes by Backprop with FGE as a function of training epoch. Red circles represent the time when network weights are saved.

As Figure 20 shows, when training the BayesianLeNet-5 model with FGE, the train loss and the prediction accuracy on the validation set performs as usual in the pre-training period because a monotone cosine annealing learning rate is used in this period. After the pre-training period ends, the model has found a point of local optima. When the training process comes into the FGE period, both the train loss curve and the

prediction accuracy curve fluctuate cyclically because a linear piecewise cyclical learning rate is used in this period. Every time the learning rate decays to the minimum value, a new point of local optima would be found and saved, which corresponds to a high-performing member model. Hence, in the FGE period, the model is walking around among different points of local optima. After the FGE period ends, all member models gather into an ensemble model. Eventually, the prediction accuracy of this ensemble model on the test set is 89.76%.

4) SWA

When applying SWA to the training algorithm, the train loss gradually decreases and the prediction accuracy gradually increases in the pre-training period because a monotone cosine annealing learning rate schedule is used in this period (see Figure 21). However, when the learning rate is switched to a high constant value, the training process comes into the SWA period and the SWA averages are formed during this period. We can see that, in the SWA period, the BayesianLeNet-5 model with averaged network weights shows great performance. The train loss curve and the prediction accuracy curve stabilize at a high-performing level. Eventually, the prediction accuracy of the BayesianLeNet-5 model with SWA is 89.77%.

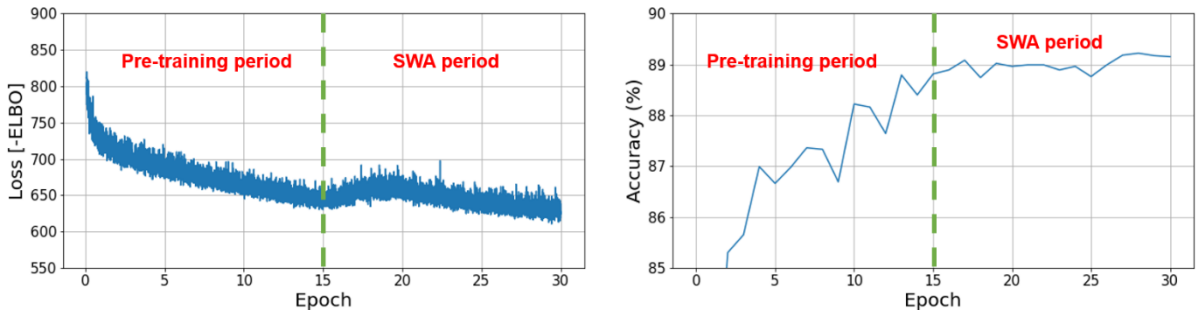

Figure 21. Left: The train loss of the BayesianLeNet-5 model trained by Bayes by Backprop with SWA as a function of training epoch. **Right:** The prediction accuracy of the BayesianLeNet-5 model trained by Bayes by Backprop with SWA as a function of training epoch. Red circles represent the time when network weights are saved.

5) Summary

Finally, training algorithms with different ensemble learning strategies are benchmarked and the performance of their output models is evaluated, and the results are shown in Table 2 and Figure 22. We can get some conclusions from it:

(1) Compared with the standard single BayesianLeNet-5 model trained using Bayes by Backprop, all ensemble models have better generalization performance on the test set. It indicates that applying ensemble learning strategies in the BNNs training algorithms is exactly very helpful to improve the performance of models.

(2) In this work, the ensemble model trained with the traditional ensembling strategy shows the best prediction accuracy on the test set. This might be because member models in this ensemble are trained independently so there exists a strong difference among them. In other words, the solutions of local optima found by different member models are sufficiently far away from each other on the loss surface.

(3) However, the training time taken by the training algorithm with traditional ensembling is the longest because the total number of training epochs it requires is the largest.

(4) The second slowest is the training algorithm with Snapshot Ensembling as the cycle length in Snapshot Ensembling is relatively large (30 epochs).

(5) FGE outperforms Snapshot Ensembling. As the cycle length in FGE is much shorter (4 epochs), it takes less training time than Snapshot Ensembling. Furthermore, the ensemble model trained with FGE has better generalization performance than that of Snapshot Ensembling.

(6) Compared with other ensemble learning strategies, the most significant advantage of SWA is that it can provide a great prediction accuracy, which is comparable to FGE, without costing too much additional training time.

Table 2. Comparison of training time and prediction accuracy among different ensemble learning strategies.

Training Method	Training Time (s)	Prediction Accuracy on the Test Set (%)
Standard Single Model (control)	550	87.25
Traditional Ensembling	5667	91.48
Snapshot Ensembling	5576	88.91
FGE	3215	89.76
SWA	857	89.77

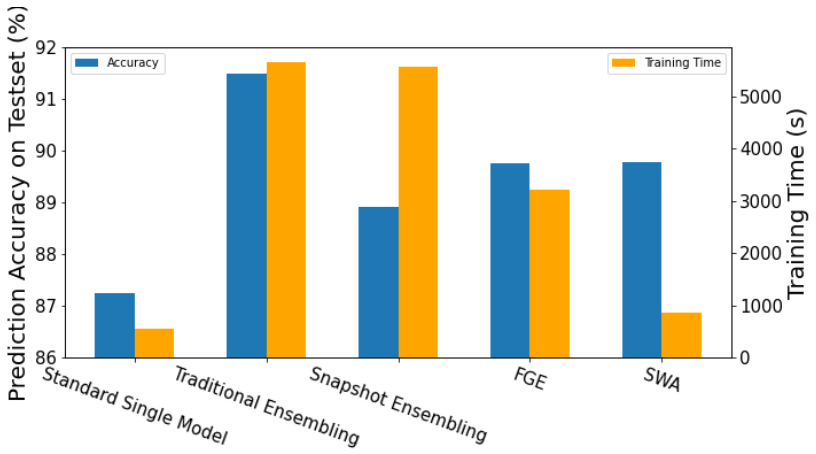

Figure 22. The combined bar chart of prediction accuracy of models trained with different ensemble learning strategies on the test set (blue bar) and the total training time for different ensembling algorithms (orange bar).

V. CONCLUSION

There are two works implemented in this dissertation.

In the first work, different Bayesian Inference algorithms are used to train BNN models based on the architecture of the MLP network. The experimental results demonstrate that, compared with traditional point estimate ANNs, BNNs have better generalization performance on the test set. This is because the network weights of BNNs are probability distributions rather than single fixed values, which makes BNN models take into account as many possible weights as possible when they do predictions. In addition, the ability of BNNs to quantify the uncertainty associated with their predictions is also tested. For an input image that is not easy to be classified, BNN models can express the uncertainty associated with their predictions through prediction histograms. Furthermore, we can set a rule in the prediction algorithm of BNNs, which can make BNN models refuse the request for prediction in some situations, but the specific condition of this rule should depend on the circumstances. For example, in terms of technologies that get involved in safety problems like self-driving cars, we can set a relatively strict rule in its decision-making system. By doing so, human users can take back control when they encounter some tough problems which ANNs cannot deal with. It can effectively reduce the risk of potential accidents. Therefore, BNNs can not only provide more reliable outputs but also say 'I don't know' when they are not confident enough about their decisions.

In the second work, a variety of ensemble learning strategies are applied in the training algorithm of BayesianLeNet-5. The experimental results indicate that all ensemble learning strategies used in this work are helpful to improve the generalization performance of BNN models. It demonstrates that ensemble learning strategies that were designed for training traditional point estimate ANNs can still work out in training BNNs. In addition, in order to benefit from ensemble models, traditional ensembling, Snapshot Ensembling, and FGE have to afford the price of consuming more computing resources and training time because they need to store multiple member models. However, SWA can provide a great performance improvement without too much additional computational overhead. Therefore, SWA is an ensemble learning strategy of high performance-to-cost ratio.

Overall, BNNs can overcome some inherent problems existing in traditional point estimate ANNs, and many ensemble learning strategies can further improve the performance of BNNs. In the future, the demand for artificial intelligence will not be confined to very high prediction accuracy. We should pay more attention

to dealing with situations where artificial intelligence makes mistakes. After all, people inevitably make mistakes in daily life no matter how intelligent they are, and so does artificial intelligence.

REFERENCES

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, (1989). "Backpropagation applied to handwritten zip code recognition", *Neural Computation*, **1**(4):541-551.

A. Krizhevsky, I. Sutskever, and G. E. Hinton, (2012). "Imagenet classification with deep convolutional neural networks", *Communications of the ACM*, **60**(6): 84-90.

E. Goan, and C. Fookes, (2020). "Bayesian neural networks: An introduction and survey", in *Case Studies in Applied Bayesian Data Science*, pp. 45-87.

L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, (2022). "Hands-on Bayesian neural networks—A tutorial for deep learning users", *IEEE Computational Intelligence Magazine*, **17**(2): 29-48.

A. G. Wilson, and P. Izmailov, (2020). "Bayesian deep learning and a probabilistic perspective of generalization", *Advances in neural information processing systems*, 33, 4697-4708.

R. M. Neal, (2011). "MCMC using Hamiltonian dynamics", *Handbook of Markov chain Monte Carlo*, **2**(11): 2.

A. D. Cobb, and B. Jalaian, (2021). "Scaling Hamiltonian Monte Carlo inference for Bayesian neural networks with symmetric splitting", in *Uncertainty in Artificial Intelligence*, pp. 675-685.

G. E. Hinton, and D. Van Camp, (1993). "Keeping the neural networks simple by minimizing the description length of the weights", in *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pp. 5-13.

K. Friston, J. Mattout, N. Trujillo-Barreto, J. Ashburner, and W. Penny, (2007). "Variational free energy and the Laplace approximation". *Neuroimage*, **34**(1), 220-234.

T. S. Jaakkola, and M. I. Jordan, (2000). "Bayesian parameter estimation via variational methods", *Statistics and Computing*, **10**(1): 25-37.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, (2015). "Weight uncertainty in neural network", in *International conference on machine learning*, pp. 1613-1622.

G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, (2017). "Snapshot ensembles: Train 1, get m for free", *arXiv preprint arXiv:1704.00109*.

T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson, (2018). "Loss surfaces, mode connectivity, and fast ensembling of dnns", *Advances in neural information processing systems*, **31**.

P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, (2018). "Averaging weights leads to wider optima and better generalization", *arXiv preprint arXiv:1803.05407*.

N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, (2016). "On large-batch training for deep learning: Generalization gap and sharp minima", *arXiv preprint arXiv:1609.04836*.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, (1998). "Gradient-based learning applied to document recognition". *Proceedings of the IEEE*, **86**(11): 2278-2324.