Stochastic Gradient Descent as Approximate Inference

with Application on Distributed Bayesian Learning

Raj Kiriti, Katherine Tsai, and Chase Duncan

University of Illinois Urbana-Champaign

Abstract

We propose Distributed stochastic weight averaging Gaussian (DSWAG) in the distributed deep learning setting. This method reduces the computation cost of approximating the posterior distributions of Bayesian neural networks and generalizes well on testing data. The generalization can be explained by observations of the geometric landscape of the loss functions. Meanwhile, we study the background of approximate inference and the connection of SWAG with stochastic calculus.

1 Introduction

Bayesian Neural Networks (BNN) encapsulate weights in distributions rather than the point estimates of standard neural networks. Such probabilistic modeling retains model uncertainty, is robust to overfitting [1], and provides a confidence interval for inference. Applications of BNN range from autonomous driving to language processing to medical diagnosis [2, 3]. However, computing the posterior distributions of BNN is expensive. In general, there are no analytical solutions due to the nonlinear structures of neural networks. Therefore, we resort to numerical methods to approximate the posterior distributions. Stochastic weight averaging Gaussian (SWAG) [4] is a simple and scalable method to approximate the posterior distributions of BNNs using stochastic gradient descent (SGD). In addition to its low-computation cost, this method generalizes well on testing data. The better generalization is attributed to finding wide flat local minima. This is done by computing Polyak–Ruppert averaging [5] of SGD estimates. In this project, we develop and analyze an algotithm which distributesthe SWAG method, further reducing the computational cost of BNNs.

In general, distributed learning seeks to lower computation cost by partitioning work and distributing it to several workers, which may be a network of machines or multiple processors on a single machine. Since each worker only sees a subset of the training data, a synchronizing step is required to update the model. This is generally done by taking the average gradients sent by workers and then sending the updates back to workers [6]. We explore a similar strategy for distributed learning of the SWAG algorithm.

In section 2, we discuss the history of BNNs and the geometric landscape of neural networks. In section 3, we discuss the algorithm of SWAG method, its interpretation in the perspective of the geometric landscapes and connection to stochastic calculus. In section 4, we discuss the distributed learning settings. In section 5, we discuss the algorithm of distributed-SWAG and in section 6, we will show some experiment results. Finally, in section 7, we will conclude the proposed framework and discuss future work.

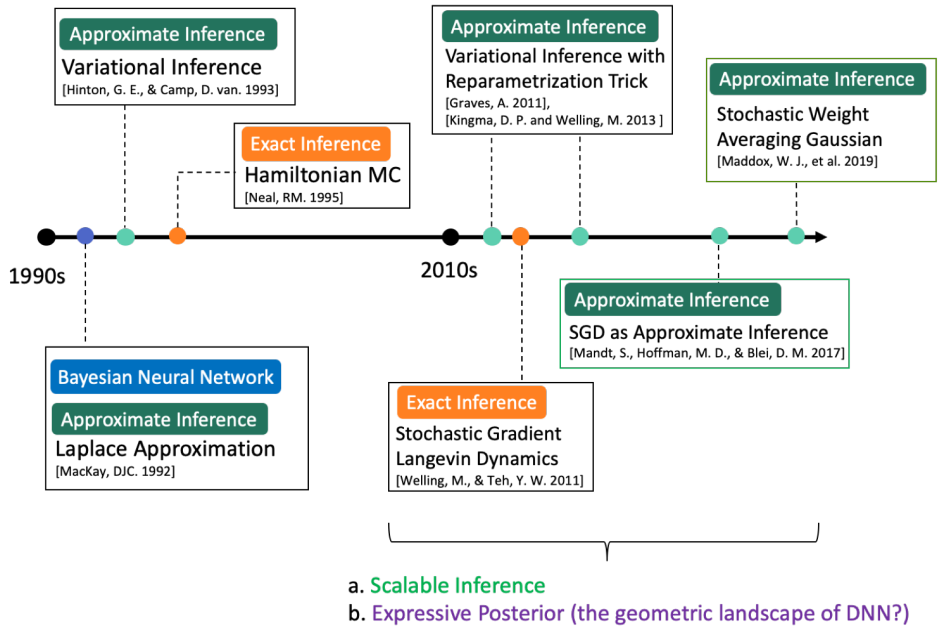

Figure 1: Timeline of the development of BNN and associated approximate inference methodologies discussed in this project

2 Prior Work

2.1 History of Bayesian Neural Network and Approximate Inference

The origin of Bayesian neural network dates back to 1992 when David Mackay published a seminal work on the model structures of Bayesian neural network [7]. The proposed algorithm uses the Laplace approximation method to estimate the posterior. Subsequent work includes variational inference [8] and Markov chain Monte Carlo (MCMC) [9]. These methods are not scalable to large datasets. For example, the Hamiltonian MCMC requires computing the full gradient and the variational inference is carried out by expectation maximization algorithm. As a result, work on BNNs remained largely dormant until the early 2010 when multiple new works brought about a revival of BNNs. Several works propose algorithms that make inference scalable to large datasets. Two exemplary cases are the stochastic gradient Langevin dynamics (SGLD) [10] and reparametrization trick of variational inference [11]. SGLD is a stochastic gradient descent version of MCMC. SGLD converges to the true posterior distribution asympototically with decreaing learning size. The reparameterization trick of variational inference reparameterizes the stochastic function into the sum of deterministic term and stochastic term. This allows us to move the gradient operator inside the integration. The integration can then be approximated using Monte Carlo sampling.

Contrary to the stochastic MCMC that converges to the exact posterior distribution asymptotically, SWAG uses SGD updates with a constant learning rate to approximate the posterior distribution. As we'll see shortly, this method is fairly simple and generalizes well. Before looking at the algorithm, we first briefly discuss the geometric landscape of the posterior distribution, which helps us better understand how to construct a good inference algorithm.

2.2 Geometric Landscape and Better Generalization

Several prior works [13, 14, 15] suggest that sharp local optima have poor testing accuracy while flat local optima have a more reasonable testing accuracy. From the information theoretic perspective, the minimum description length (MDL) explains that statistical models which require fewer bits to describe generalize better [15]. Empirically, the SGD estimate converges at the rim of the local basin, where there is a sharp cliff on the other side. Therefore, the estimate is stuck and can not escape. However, this also implies that the SGD estimate has poor generalization. Taking the average of SGD estimates are shown to move the estimate to the center of the local basin, where the geometric landscape is flat as figure 2 shown. This leads to better generalization in testing.

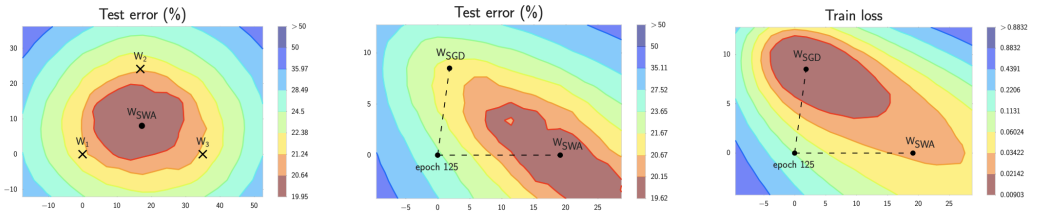

Figure 2: (figure from [12]) The geometric landscape of SWA method. **Left**: W_1, W_2, W_3 are three independent estimates and W_{SWA} is the average of three estimates. **Center, right**: The geometric landscape of the testing data v.s. the geometric landscape of the training data. While W_{SGD} has better performance on the training data, W_{SWA} generalizes better on testing data.

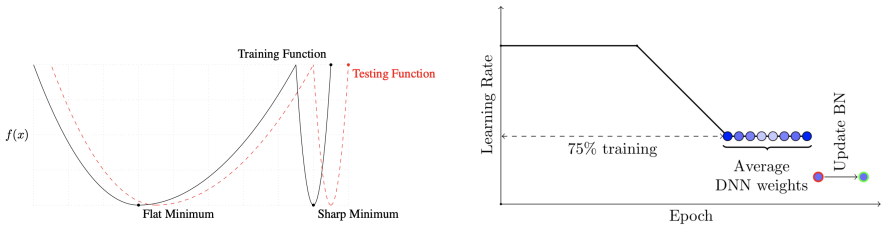

Figure 3: **Left**: (figure from [15])The black line indicates the loss on training data while the red dashed line indicates the loss on testing data. **Right**: (figure from [12]) The training procedure of the SWA method.

In fact, taking averaging of stochastic gradient descent is not new and is often referred as the Polyak-Ruppert averaging [5]. This method is also known to be more robust to noise in gradient estimates. However, taking the average of previous iterations also implies slow forgetting of the initial estimates. In practice, a common adaptive approach is to take the average of SGD estimates at the last few iterations. The stochastic weight averaging (SWA) algorithm [12] adopts this approach and divides the training session into two sections: the first 75% of the training session is used to find a good local region with decreasing learning rate and the last 25% of the time is used to find the center of a flat local minima by taking average of gradient estimates shown in figure 3. After allocating flat local optima, the following step is to estimate the width of the local basin, namely, the variance of the posterior distribution.

2.3 Distributed learning

[16] suggest the use of a posterior server architecture for the distributed Bayesian learning where they use stochastic natural gradient based expectation propagation for inference. In their architecture each worker has a subset of data and calculates a likelihood from it, in addition they maintain a cavity distribution which is essentially the conditional distribution over the parameters given data on all other workers. Each worker then communicates asynchronously with the posterior server sending their current likelihood estimate and receiving a new cavity distribution. [17] propose a more robust posterior estimation using median. They construct Gaussian posteriors on each device and use *Weiszfeld's algorithm* to calculate the median of the obtained distributions. This robust estimate can provably converge if the number of byzantine devices is less than half. A more similar line of work to the one proposed is [18]. They use a similar idea from [12] but in a distributed manner. However their approach is to optimise for better generalisation and doesn't fall under the category of Bayesian Inference.

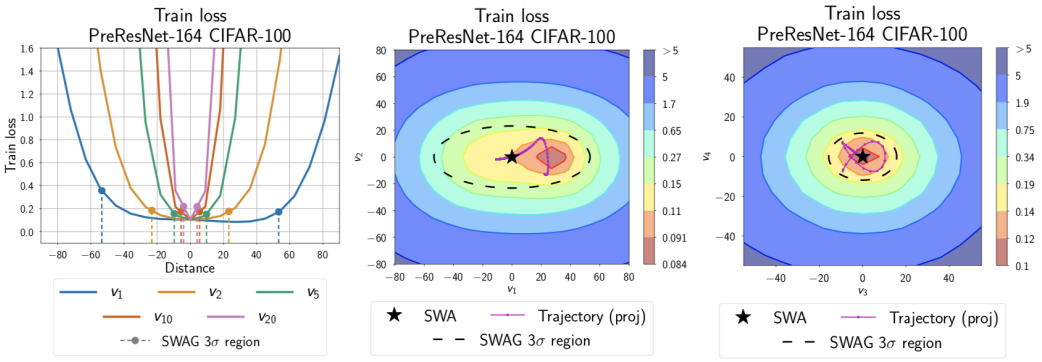

Figure 4: (figure from [4]) **Left**: Train loss along the direction of the eigenvectors of the posterior covariance **Middle**: Posterior joint density surface in the plane spanned by eigenvectors of SWAG covariance matrix corresponding to the first and second largest eigenvalues and **Right**: the third and fourth largest eigenvalues.

3 SWAG and Connection with Approximate Inference

In this section we will discuss how to compute the second moments the posterior distributions and interpretation of this algorithms in the perspective of stochastic calculus.

3.1 Covariance Structures

The SWAG constructs the diagonal and low-rank structure of the posterior covariance matrices. After sampling S models in the local region, we compute the variance of S samples. In addition, the low-rank structure is computed by the sample covaraince of last K samples.

3.2 The Geometric Explanation

In the previous section we state that a good local optimum has a flat geometric landscape. To demonstrates that the SWAG algorithm finds a flat region, we can do the eigen-decomposition of the covariance matrix and look at the geometric landscape along the direction of the eigenvector shown in figure 4. The figure shows that SWAG captures the flat low-dimensional geometry of the posterior distribution. However, since the model is low-rank, it will underestimate the covariance along random directions.

3.3 SGD as Approximate Inference

Constant SGD can be viewed as a stochastic process with stationary distribution. In particular, we can approximate the constant SGD estimates with a continuous-time Ornstein-Uhlenbeck process [19]. The Ornstein-Uhlenbeck process has a desirable property that its stationary distribution is a Gaussian distribution. Consequently, the constant SGD can approximate the posterior distribution well if it is constrained in a flat local region that can be approximated by quadratic function. Moreover, it has also be shown that weight averaging has optimal convergence rate and can not be improved by any preconditioning matrix [19].

4 Distributed learning

Consider the problem where there are n devices, with private data on each device which cannot be shared directly, the goal is to train a global model with this data without being shared explicitly. This can be formulated as

$$\min_{\theta \in \mathbb{R}^d} F(\theta) \quad \text{where} \quad F(\theta) = \frac{1}{n_s} \sum_{i \in \mathcal{S}} \mathbb{E}_{z_i \sim \mathcal{D}_i} f_i(\theta, z_i) \tag{1}$$

Where $\mathcal{D}_i$ is the data on the ith device and z_i is sampled from that data. For general machine learning problem we take $f_i = l(x_i, y_i, \theta)$ which is the loss of the prediction function for example (x_i, y_i). If the partitioning of data is done uniformly across the devices which is the case for an independent identical distribution (IID) then $\mathbb{E}_{z_i \sim \mathcal{D}_i} f_i(\theta, z_i) = \mathbb{E}_{z_j \sim \mathcal{D}_j} f_j(\theta, z_j)$. In a general Federated learning setting[20], the data is distributed in a non-IID manner. Similar to federated learning to minimize redundancy and communication costs we consider a case of on device learning by letting each device optimize the loss function on it's data using SGD.

5 Distributed SWAG

We propose DSWAG: SGD with distributed bayesian inference. We divide DSWAG training into three phases (as in algorithm 1). In the first phase there is a synchronized training where each client device runs stochastic gradient descent locally for an arbitrary number but same number of epochs on each device. A high learning rate is used for this phase. During the second phase each client device independently updates its model and a cyclic learning rate is used to train the models. Each device would have different models at the end of this phase. Finally during the third phase, running mean and variation are calculated which act as parameters for the Gaussian posterior.

In Phase 1, after the fixed number of iterations all the client devices send their model weights to the server, where an update algorithm, which in our case can be a simple averaging, modifies the global model and sends back the updated model to each of the devices. We also consider case where only a fraction of workers are sampled as is the case with general federated setting. This is repeated for a given number of global epochs, which is a hyper-parameter and can be tuned accordingly, but it should be made sure that the model is near the basin of attraction [12]. Stopping early might lead the individual models in phase 2 to explore different optima, while longer phase 1 might decrease the generalisation performance.

During Phase 2, the cyclic learning rate allows model to explore the search space of the weights. Number of client devices on which this phase is run can also be adjusted, with higher number leading to more exploration at the trade-off of computation and reliability of those devices. Other exploration methods like multiple cycles per device or sampling multiple times per each device can also be used for this phase.

Using the weighs obtained from the end of phase 2 at each device are used for computing Polyak averaging to compute mean and variance of the weights. As in [4] we also construct a low rank structure using the K coloumns to construct a deviation matrix $D_i = \theta_i - \bar{\theta}_i$.

Since we use a Bayesian inference, which marginalizes over the distribution of posteriors of θ, Like [4] we use Gaussian approximation of posterior from SGD iterates (as in algorithm 2)and sample from it to perform Bayesian model averaging.

This DSWAG is a step towards Bayesian inference in distributed and federated learning and hence can be used with other client and server update methods like momentum based methods or with robust aggregation like trimmed mean to decrease the model uncertainty.

6 Experiment

In this section we test the proposed algorithm on MNIST and CIFAR10 dataset. With MNIST we use a multilayer perceptron(MLP) which has one hidden layer and relu activation followed by a softmax layer. We also test MNIST with a convolutional neural network(CNN) with two convolution and pooling layers respectively, followed by a softmax layer similar to the one used in [20]. While on CIFAR10 dataset we use VGG16 [21]. For all the methods we consider we recport the test accuracy and negative log likelihood. In addition we also report the reliability diagrams following a similar approach to [4] to calibrate the uncertainty of estimates, where we plot confidence against the difference between confidence and prediction accuracy. More specifically, we split test data into multiple bins (20 in our case) uniformly using the confidence as the criterion and confidence and mean accuracy is evaluated on data from each bin. Ideally for a well calibrated model, the confidence and accuracy should take similar values. Hence values from each of the bins should lie close to zero.

Algorithm 1 `Distributed SWAG Training`

Initialize:
  $W \leftarrow$ `Number of Workers`, $\theta_0 \leftarrow$ `Initial weights`
  $K \leftarrow$ `Low rank parameter`
  $\eta_{l_1}, \eta_{l_2} \leftarrow$ `Learning rate` for `Phase 1,2`
  $T \leftarrow$ `# Total Global rounds`
  $L, M \leftarrow$ `# local client epochs for each round for phase 1,2`
Phase 1
for $t = 0, \ldots, T-1$ **do**
  Sample subset $\mathcal{S}$ of clients
  $\theta^t_{i,0} = \theta_t$
  for each client $i \in \mathcal{S}$ **in parallel do**
    for $l = 0, \ldots, L-1$ **do**
      Compute an unbiased estimate $g^t_{i,l}$ of $\nabla F_i(\theta^t_{i,l})$ from the data available
      $\theta^t_{i,l+1} \leftarrow \mathsf{Update}(\theta^t_{i,l}, g^t_{i,l}, \eta_{l_1}, t)$
    end for
  end for
  $\theta_{t+1} \leftarrow \mathsf{Update}(\theta^t_{i,l}) \ \ \forall i \in \mathcal{S}$
end for
return θ_T
Phase 2
Sample subset $\mathcal{P}$ of clients
$\theta_{i,0} = \theta_T$
for each client $i \in \mathcal{P}$ **in parallel do**
  for $m = 0, \ldots, M-1$ **do**
    Compute an unbiased estimate $g_{i,m}$ of $\nabla F_i(\theta_{i,m})$ from the data available
    $\theta_{i,m+1} \leftarrow \mathsf{Update}(\theta^t_{i,m}, g^t_{i,m}, \eta_{l_2}, t)$
  end for
end for
We get different models on each device at the end of phase 2
Phase 3
for each client update $i \in \mathcal{P}$ **do**
  $\bar{\theta} \leftarrow \frac{n\bar{\theta}+\theta_i}{n+1}, \quad \bar{\theta^2} \leftarrow \frac{n\bar{\theta}+\theta_i^2}{n+1} \quad \{moment \quad updates\}$
  if $i > |\mathcal{P}| - K$ **then**

    `Append_Col`(D,$\theta_i - \bar{\theta}$)
  end if
end for
return $\bar{\theta}$, $\Sigma_{diag} = \bar{\theta^2} - \bar{\theta}^2$ and D

Algorithm 2 `Distributed SWAG Testing`

for i=1,2..S **do**
  Draw $\tilde{\theta}_i \sim \mathcal{N}(\theta, \frac{1}{2}\Sigma_{diag} + \frac{DD^T}{2(K-1)})$
  Update batch norm statistics
  $p(y^*|Data) += \frac{1}{S} p(y^*|\tilde{\theta})_i$
end for
return $p(y^*|Data)$

Further for phase 2, Which is run on a fraction of workers selected, is an asynchronous SGD on each of the clients with cyclic learning rate. We use the following learning rate schedule

$$\alpha(t) = (1 - \frac{t}{c})\alpha_1 + \frac{t}{c}\alpha_2$$

Where t represents the epoch number and c represents the total epochs in phase 2. We set this to 20 epochs. α_1, α_2 are two constant learning rates which can be adjusted. Number of workers sampled in second phase is a hyper-parameter which can be tuned accordingly.

We run experiments for both low rank full DSWAG and DSWAG diagonal. During the testing phase, we sample from the constructed Gaussian posterior. Where the number of samples used for Bayesian inference is again a parameter that can be varied.

6.1 MNIST

For MNIST we run 100 global epochs in phase 1 with a threshold to stop when the accuracy reaches 100% since we were more interested in confidence metrics.

6.2 CIFAR 10

We repeat similar set of experiments on CIFAR 10, varying the tunable parameters

6.3 Phase 1: The Distributed Global Learning

We evenly distribute the training data on workers and train on different number of worker: $[1, 5, 10, 30, 50]$ for MNIST dataset and $[1, 2]$ for CIFAR10 dataset. We found that as we increase the number of the worker, it takes more epochs to reach the preset goal (accuracy), as the figure 5 shows. Furthermore, we plot the reliability diagram on CIFAR10 data in figure 5. The horizontal axis represents the confidence level and is computed by the maximum value of the softmax output. The vertical axis represents the difference between accuracy and the confidence. If the difference is positive, it means overconfident predictions. On the other hand, if the difference is negative, it means under-confident predictions. It can be seen from the plot that with distributed SWAG, we get more confident predictions than a vanilla SWAG algorithm.

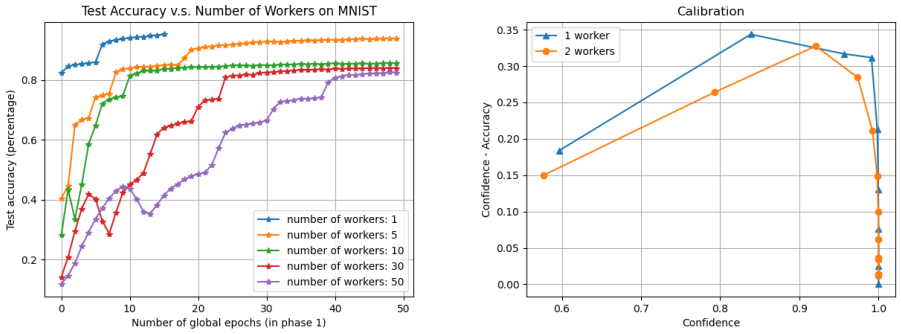

Figure 5: **Left**:When there is only one worker (blue line), the training converges fast. As we increase the number of workers, the training converges slower and reaches to suboptimal solutions. **Right**: The orange line is slightly closer to the $y = 0$ than the blue curve. This implies that increasing the number of workers slightly improve the calibration curve

6.4 Phase 2: The Geometric Landscape of Local Samplings

In this section, we demonstrates that the SWAG finds flat optima by plotting the test accuracy with the distance from the model center (model mean). As the figure 6 shows that, the testing accuracy remains almost flat as we move 100 times of standard deviation away from the center. However, as we increase the number of workers, the performance drops. We can conclude that the number of workers affects the findings of good (flat) local optima.

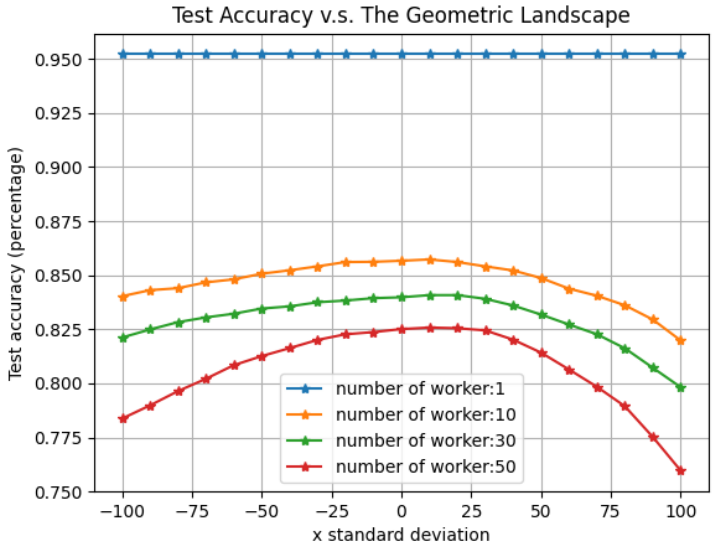

Figure 6: The x-axis represents the multiplicity of variance of the model. At $x = 0$, the points represent the MNIST testing accuracy with respect to the model mean. Similarly, at $x = 50$, the points represent the testing accuracy with respect to the [model mean]$+50\times$[model standard deviation].

6.5 Phase 3: Negative Log Likelihood

In this section, we plot the diagram of the negative loglikelihood with respect to different number of workers. For MNIST dataset, we test the number of workers $[1, 5, 30, 50]$. For CIFAR10 dataset, we test the number of workers $[1, 2]$.

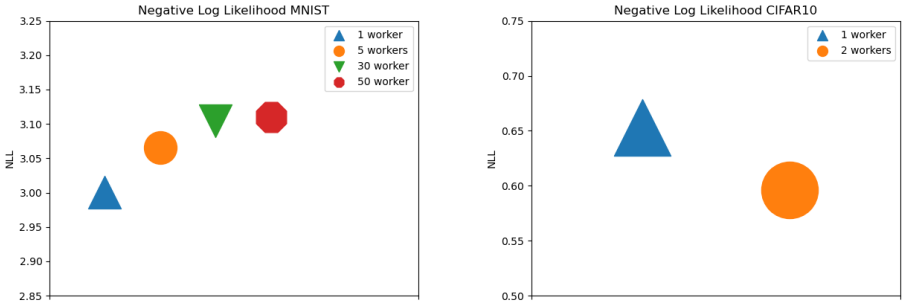

Figure 7: **Left**: For MNIST dataset, as we increase the number of workers, the negative loglikelihood increases **Right**: On the other hand, for CIFAR10 dataset, the netative loglikelihood slightly decreases as we increase the number of workers.

7 Conclusion and Future Work

We proposed DSWAG, a distributed version of Stochastic Weight Averaging- Gaussian to improve generalization and as a step towards Bayesian learning in distributed and federated setting. Our algorithm uses SGD as approximate inference. After a synchronous training phase we launch asynchronous individual training on devices with cyclic learning rate and use the models obtained at the end for constructing an approximate posterior, during the testing phase we sample from this posterior and perform Bayesian marginalization to get the final output.

We observe that, with distributed training, accuracy decreases or rather it takes higher number of epochs for the model to reach good accuracy, this is intuitive as each device gets to see only a part of the data. We further note that increasing batch size above a certain threshold might lead to decrease in performance. During the second phase if we use more clients to construct the posterior we get more confident models than when we use fewer client devices.

Further we see there is a decrease in accuracy as we increase the number of workers for distributed training, with DSWAG, we get models which are better calibrated. Moreover, we see that even with as we increase the number of workers, we still get good calibration for the models. We compare our model with the baseline experiment of SWAG after distributed training and we observe that DSWAG has better calibration performance. Our method has many hyper-parameters which can be tuned and we analyze few results with different values for these parameters, selecting optimum values for these parameters could be a direction of future work.

References

[1] Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference, 2015.

[2] Yarin Gal. Uncertainty in deep learning.

[3] Alex Guy Kendall. *Geometry and uncertainty in deep learning for computer vision*. PhD thesis.

[4] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, pages 13132–13143, 2019.

[5] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[6] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, 2011.

[7] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[8] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

[9] Radford M Neal. *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD thesis, University of Toronto, 1995.

[10] 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.

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

[12] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[13] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8789–8798, 2018.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[15] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[16] Leonard Hasenclever, Stefan Webb, Thibaut Lienart, Sebastian Vollmer, Balaji Lakshminarayanan, Charles Blundell, and Yee Whye Teh. Distributed bayesian learning with stochastic natural-gradient expectation propagation and the posterior server, 2015.

[17] Stanislav Minsker, Sanvesh Srivastava, Lizhen Lin, and David B. Dunson. Robust and scalable bayes via a median of subset posterior measures, 2014.

[18] Vipul Gupta, Santiago Akle Serrano, and Dennis DeCoste. Stochastic weight averaging in parallel: Large-batch training that generalizes well. In *International Conference on Learning Representations*, 2020.

[19] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907, 2017.

[20] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2016.

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