Optimization in Deep Neural Network

Liang Wang

April 19, 2021

Abstract

In this project, I summarize the popular optimization methods used in deep learning including two most recent deep learning researches: stochastic weight averaging **(SWA)**[Timur Garipov and Dmitry Vetrov, 2018a] and stochastic weighted average gaussian **(SWAG)** [Maddox, 2019]. Specifically, with some mathematical derivation, I discussed intuitively how to maintain numerical stability using batch normalization[Ioffe and Szegedy, 2015] and weight initialization[Simone Rossi, 2018]; how to overcome the dimension varied optimization oscillation using Adagrad[Duchi and Singer, 2011], RMSprop[Tieleman, 2012] and Adam[Diederik P. Kingma, 2015] and their connection to proximal gradient optimization; how does the **SWA** help to improve on multimodal generalization and how is the **SWAG** used to provided bayesian neural network solution. To explore those optimization methods, I implemented Adam, **SWA** and **SWAG** on the famous [CIFAR10] dataset without resorting to any built-in pytorch optimization packages. The computation was conducted on Boston University Shared Computing Cluster [BUSCC] after parallelization with code available on github. My result indicates the effectiveness of **SWA** in finding a "flat minimum" [Nitish Shirish Keskar and Tang, 2017] and the effectiveness of the **SWAG** in providing reasonable prediction confidence interval.

1 Introduction

Due to its successful application in the modern society, Deep Neural Network(**DNN**) has gained its well deserved attention across both academia and industry. Since the last three decades, the research has evolved rapidly and has been in a close connection to the optimization theories. In general, the neural network in sample training can be understood as a function approximation process with elegant numerical tricks to stabilize and accelerate the optimization. In recent researches, the process has been combined with many interesting statistical concepts, which have close connection to stochastic optimization. In particular, it can be shown that when the number of neuors and number of hidden unit increases to infinity, the whole function training converges to a kernel regression with gaussian process as the function prior[Blake Bordelon, 2020]. [Stephan M, 2017] showed that under certain conditions, the stochastic gradient decedent converges to a gaussian SDE, whose stationary distribution could be obtained and employed for stochastic sampling. [Maddox, 2019] utilize the connection between the stationary SDE and the stochastic gradient optimization path to propose moment estimator on the posterior weight covariance. Besides, there are also researches that concentrate on generalizing the neural network structure for specific machine learning tasks. For example, [Felix A. Gers, 1999] the LSTM was utilized in time series modeling and has been enriched with bidirectional network to learn human language under context [Schmidhuberab, 2005]. The CNN [Y. Le Cun and Jackel, 1999] has been successfully employed for pattern recognition. Here in this project, I summarize the optimization problem occurred in neural network in the past decades and, with the CIFAR10 pattern recognition dataset as an example, I implement the stochastic weight averaging **(SWA)** and stochastic weight averaging gaussian **(SWAG)** [Maddox, 2019] along with Adam optimization[Diederik P. Kingma, 2015].

2 Optimization Problem

In this section, I summarize the bottlenecks encountered in recent neural networks researches and the corresponding paper to solve the problem.

2.1 Problem Setup

Without loss of generality, for an example of multilayer classification perceptron, it consists of hidden neuros, hidden layers and activation functions as indicated in Fig 1.

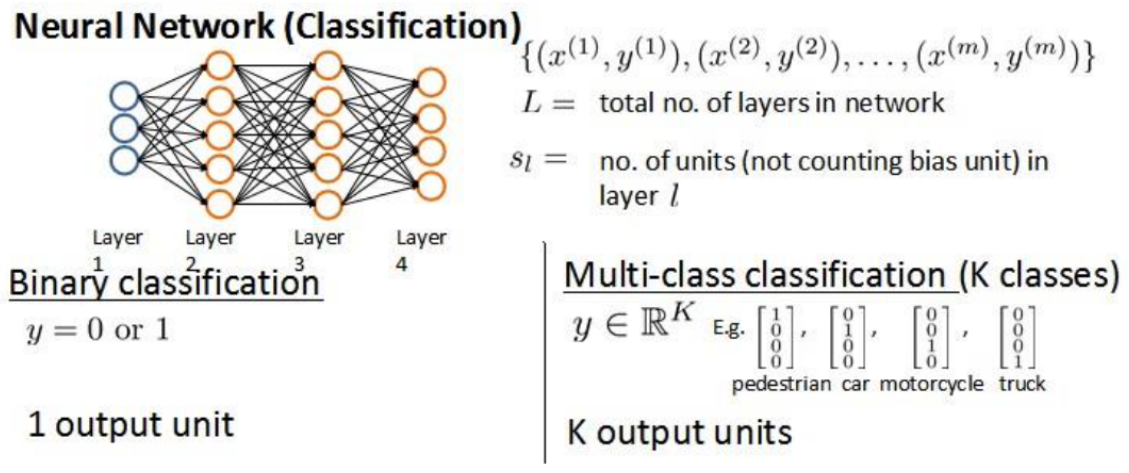
Neural Network (Classification)
$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
$L =$ total no. of layers in network
$s_l =$ no. of units (not counting bias unit) in layer l
Layer 1
Layer 2
Layer 3
Layer 4
Binary classification
$y = 0$ or 1
1 output unit
Multi-class classification (K classes)
$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$
pedestrian car motorcycle truck
K output units

Figure 1: **Neural Network Structure** *[Ng, 2012]*

The problem in general can be considered as approximating some $Y = f(X)$ relationship by using observed data (X, Y). The model setup in Fig 1 suggests the function relationship $f(x)$ depends on specific choice of weight w, which are considered as the main parameter used to optimize the distance between the approximating function and the true function. Denote $\Theta = w$, the classification problem then aims to solve for $\text{argmin}_\Theta E[J(\Theta)|X]$ with $J(\Theta)$ given as follow:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{k} y_k^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-\left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\Theta_{ji}^{(l)}\right)^2$$

The problem is non-convex and usually overparameterized. The gradient calculation employ the chain rules layer by layer and thus permit a computational graph representation. The computation graph together with the automatic differentiation together function as the cornerstone of deep learning research. Both of the methods are well implemented in modern software, however, we will still need to be aware of the optimization issues summarized as follow:

2.2 Unstable Gradient

Since the gradient computation employ the chain rule layer by layer, the gradient computation of the first layer is in a product relationship w.r.t the later layers (Eq 1). When the neural network gets deeper, the gradient computation suffers from the gradient exploding and gradient vanishing. Specifically, suppose at iteration t, there are in total d hidden layers

$$\frac{\partial L(\cdot)}{\partial W_t^1} = \frac{\partial L(\cdot)}{\partial h_t^d}\frac{\partial h_t^d}{\partial h_t^{d-1}}\cdots\frac{\partial h_t^1}{\partial W_t^1} \tag{1}$$

One simple solution to the gradient exploding is to cap the gradient of each weight update $\frac{\partial h_t^d}{\partial W_t^d}$ by a constant c or using $\frac{\partial h_t^d}{\partial W_t^d}||\frac{\partial h_t^d}{\partial W_t^d}||^{-1}c$ to substitute the gradient update, which is called gradient clipping or norm clipping. The result was theoretically justified to be effective in [Razvan Pascanu, 2013]. However, such a method is only effective in preventing gradient exploding. With a general idea of normalizing the gradient into an appropriate scale, there are two other popular methods proposed to tackle this problem.

2.2.1 Bach Normalization

The idea of batch normalization [Ioffe and Szegedy, 2015] is to re-scale and re-center of the neural network so that the product of the gradient backpropagation is within reasonable computer precision. For a specific mini-batch and hidden layer d with m output $(h_1^d \cdots h_m^d)$, it conduct the following operation $\forall i = 1...m, \forall$ hidden layer d:

$$\begin{aligned}\mu_b &= \frac{1}{m}\sum_{i=1}^{m} h_i^d \\ \sigma_b^2 &= \frac{1}{m}\sum_{i=1}^{m}(h_i^d - \mu_b)^2 \\ \tilde{h}_i^d &= \gamma^d \frac{h_i^d - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta^d\end{aligned} \tag{2}$$

where the γ^d, β^d are layer specific parameters updated through the training process and thus we are entitling the neural network with the ability to learn how to scale the the gradient appropriately through the training process.

Recently, it has been found that the batch normalization [Simone Rossi, 2018] not only aids the optimization by stabilizing the gradient, but also regularize the network. The regularization is introduced as a side-effect where the output of each layer is normalized using the data, which contains random noises. Injecting random noise to the neuron output and it can be understood as a restricting a bayesian prior, which has been proven to be effective in neural network training.

2.2.2 Smart Initialization

Besdie the batch normalization, the initialization point of optimization problem can be essential especially for nerual network optimization where gradient overshooting is very likely to happen with bad initialization point. For a specific layer i and the next layer $i+1$, denoting the weight multiplication $s^i = z^i w^i + b^i, z^{i+1}, i = 1...d-1$, the size of the layer i as n_i, [Glorot and Bengio, 2010] analyzed the variance of the gradient w.r.t s^i, w^i analytically and found:

$$\begin{aligned}
Var[\frac{L(\cdot)}{\partial s^i}] &= Var[\frac{L(\cdot)}{\partial s^d}] \prod_{i'=0}^{d} n_{i'+1} Var[W^{i'}] \\
Var[\frac{L(\cdot)}{\partial w^i}] &= \prod_{i'=0}^{i-1} n_{i'+1} Var[W^{i'} \prod_{i'=0}^{d} n_{i'+1} Var[W^{i'}] Var[x] * Var[\frac{L(\cdot)}{\partial s^d}]
\end{aligned} \tag{3}$$

Under the condition of smooth forward information flow and smooth backward information flow:

$$\begin{aligned}
&\forall (i, i'), Var[z^i] = Var[z^{i'}] \\
&\forall (i, i'), Var[\frac{L(\cdot)}{\partial s^i}] = Var[\frac{L(\cdot)}{\partial s^{i'}}]
\end{aligned} \tag{4}$$

It turns out that with Relu activation, initializing $W \sim U[-\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}]$ could maintain $Var[W^i] = \frac{2}{n_i+n_{i+1}}, \forall i$, which function as a compromise of Eq 4. There are other initialization methods such as [Simone Rossi, 2018], which runs bayesian regression between each layers to initialize the bayesian neural network weights. However, such an initialization is expensive but maybe alleviated with bayesian approximate sampling method such as INLA [Rue and Chopin, 2009].

2.3 Dimension Varied Oscillation

Another important issue is the choice of small step-size. Since the neural network model is highly ooverparameterized, its gradient magnitude are often in large diversity. As a result, adopting large learning rate would overshoot certain dimensions while adopting small learning rate would stop the learning for some important weight parameters. To tackle such an issue, there are mainly three important optimization contribution in this regard:

2.3.1 Adagrad

The Adagrad normalizes the dimension varied oscillation by using the stochastic gradient computation along the iteration. Its derivation is in close connection to the proximal gradient optimization. We know that for stochastic gradient descent, it is using the update rule $x^{k+1} = x^k - r^k \nabla f(x^k)$ with $\nabla f(x^k) \approx \frac{1}{N}\sum_{i=1}^{n} \nabla f(x_i^k)$. I turns out the update rule is also the solution of the proxmial gradient optimization:

$$\begin{aligned} x^{k+1} &= \operatorname*{argmin}_{x \in X} f(x) + \frac{1}{2r^k}||x - x^k||_2^2 \\ &\approx \operatorname*{argmin}_{x \in X} f(x^k) + \nabla f(x^k)^T(x - x^k) + \frac{1}{2r^k}||x - x^k||_2^2 \\ &= \operatorname*{argmin}_{x \in X} \nabla f(x^k)^T x + \frac{1}{2r^k}||x - x^k||_2^2 \end{aligned} \tag{5}$$

[Duchi and Singer, 2011] argued that the use of L_2 norm as the proximal regularization doesn't reflect the geometry of the optimization problem and thus proposed to use a positive definite matrix B to scale the L_2 penalty. If we define $||X||_B^2 = x^T Bx \geq 0$, and try to solve

$$x^{k+1} = \operatorname*{argmin}_{x \in X} f(x) + \frac{1}{2r^k}||x - x^k||_B^2 \tag{6}$$

With similar derivation, we could obtain $x^{k+1} = x^k - r^k B^{-1} \nabla f(x^k)$ and if we use $G_t = \sum_{i=1}^{t} \nabla_w L(w_i) \odot \nabla_w L(w_i)$ to estimate the loss geometry, we could improve our stochastic gradient update

$$r_t = \frac{\alpha}{\sqrt{\epsilon I + diag(G_t)}} \tag{7}$$

Remark Intuitively, the adagrad is using the G_t to approximate the curvature of the optimization loss space and thus appropriately scale each dimensions to ensure the update in the correct scale. But as the optimization always increases the G_t, the r_t will always decays to 0 as the iteration increases. This method thus only works for simple models in a similar manner to the weight decay method but for complicated models such as neural network, better method has been proposed.

2.3.2 RMSprop

The basic idea of RMSprop is similar to Adagrad, which uses the gradient's second moment as an estimate of the loss geometry curvature. However, instead of decreasing the learning rate in each step, the RMSprop[Tieleman, 2012] uses an exponential weighted geometry information to update the learning rate. It thus permit longer iteration with the following modified update:

$$S_{dw} = \beta_1 S_{dw} + (1 - \beta_1)(dw \odot dw)$$

$$w_{t+1} = w_t - r_t \frac{dw}{\sqrt{S_{dw} + \epsilon}}$$

Note that It is using exponentially weighted average because for $\beta_1 \in (0, 1)$, its an exponentially weighted average since $\frac{1}{1-\beta_1} = 1 + \beta_1 + \beta_1^2 + ...$

2.3.3 Adam

The Adam [Diederik P. Kingma, 2015] method can be considered as a generalization of the momentum optimization and RMSprop. It updates the weight with exponentially weighted gradient after exponentially weighted scale. It turns out that by incorporating the exponentially weighted first moment into the update consideration, not only the gradient movement is smoothed, but also the algorithm is entitled with the ability to escape from an optimization stationery point.

$$\begin{aligned} S_{dw} &= \beta_1 S_{dw} + (1 - \beta_1)(dw \odot dw) \\ V_{dw} &= \beta_2 S_{dw} + (1 - \beta_2)dw \\ w_{t+1} &= w_t - r_t \frac{V_{dw}}{\sqrt{S_{dw} + \epsilon}} \end{aligned} \tag{8}$$

Usually, $\beta_2 = 0.9, \beta_1 = 0.999$ are chosen by default and $\epsilon = 10^{-8}$ was set to maintain numerical stability. In the original paper, a biased adjusted version of the update is also available but for the bias correction doesn't matter that much in long iteration.

2.4 Multi Modal Non-convexity

In modern statistical researches, the value of the loss function is data dependent and thus can be considered as random variable. Thus not only the optimization surface of the training data needs to be studied, but also the randomness in the testing data needs to be taken into consideration. [Schmidhuber, 1997] first propose the idea of 'flat minimum', which is defined as a large connected region within the weight space where the error of the loss function maintains little variation(see Fig 2). Statistically, the flat minimum has been proven to be important for model generalization. This can be understood from both the prospective of the Minimum Description Length[Rissanen, 1983] and the prospective of the bayesian statistics. Specifically, the flat minimum requires a model with less likelihood specification compared to a sharp minimum and thus generalize better. Thanks to the recent neural network researches, we now have a better understanding of the testing loss structure. In [Nitish Shirish Keskar and Tang, 2017], it has been demonstrated that the plain SGD optimized **DNN** tends to find sharp minimum on the boundary of and thus pointed out the potential improvement for moving toward a flat minimum.

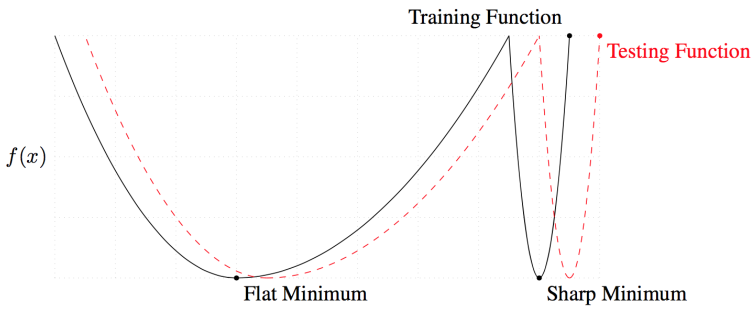

Figure 2: **Sharp minimum v.s Flat Minimum** *[Nitish Shirish Keskar and Tang, 2017]*

In [Timur Garipov and Dmitry Vetrov, 2018b], the existence of the smooth local minimum path has been justified in a subspace and thus welcome one of the most recent research in deep neural network training: the Stochastic Weighted Average **(SWA)** method [Timur Garipov and Dmitry Vetrov, 2018a]. In summary, after the model training with parameter $\hat{W}$, the **SWA** explore along the loss surface with cyclic learning rate and $\hat{W}$ as initialization to find different local optimum solutions $\hat{W}_i$. Then it averages over those local optimum solutions $\hat{W}_{swa} = \sum_{i=1}^{T} \hat{W}_i$(Fig 3), which has been proven to be within the center of the wide optimum surface and thus entitle better generalization. The idea turns out to be in close connection to Polyak-Ruppert averaging [Polyak and Juditsky, 1992] in convex optimization.

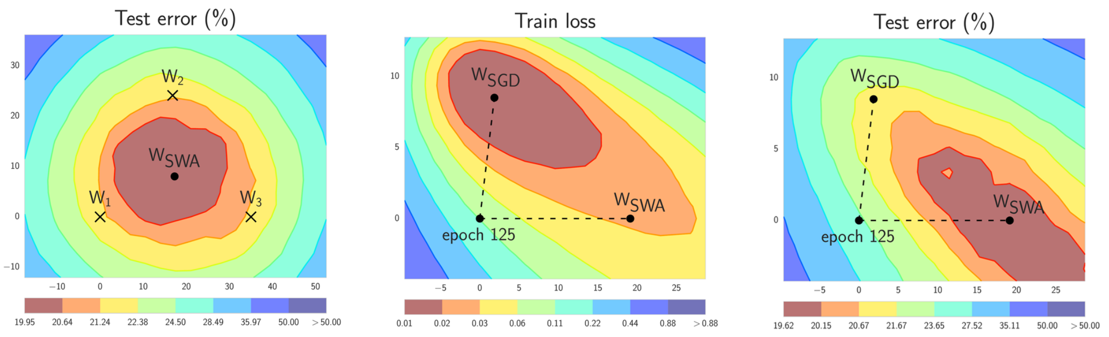

Figure 3: **SWA Loss Surface** *picture from SWA paper [Timur Garipov and Dmitry Vetrov, 2018a]*

2.5 Uncertainty Quantification

In an analog to the sensitivity analysis in traditional optimization theorems, we sometimes care not only about the optimum solution but also about the robustness of the optimum solution. The bayesian marginalization has provided an interesting prospective for the uncertainty quantification in deep neural network. Essentially, after training our model with data D and given new input X, bayes marginalization states that our prediction should be given by:

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

where $p(w|D)$ is the posterior distribution of the neural network weights, the integral is intractable in most of the case and thus a Monte Carlo estimate often comes in for approximation:

$$p(y|x, D) \approx \sum_{s=1}^{S} p(y|x, D, w_s), w_s \sim p(w|D)$$

It turns out that under the complicated structure of the neural network, $p(w|D)$ are usually unavailable. Variational inference and laplacian approximation have been traditionally used for approximation. However, recently [Stephan M, 2017] has found that, under some assumptions, the neural network stochastic gradient weight update follows the following SDE dynamic:

$$d\theta(t) = -\epsilon g(\theta)dt + \frac{\epsilon}{\sqrt{S}} B dW_t \tag{9}$$

where $d\theta(t)$ is the weight update, $g(\theta)$ is the weight gradient, ϵ is step size, B is kernel, dW_t is Brownian Motion, S is the batch size.

This derivation has provided a gaussian prospective for the posterior weight distribution with its moment information contained in the weights gradient $g(\theta)$ and the kernel B. And not surprisingly, since **SWA** can be considered as averaging out the finite difference of the weight SDE in Eq 9, the 2nd moment statistics can also be estimated using the gradient information along the **SWA** exploration. [Maddox, 2019] thus generalize the **SWA** idea to include the gaussian variance estimate as $Diag(\sum_{i=1}^{T} \frac{1}{T}(E[\hat{w}_i^2] - \hat{w}_{swa}^2))$. To save for the memory complexity, the estimate is replaced with a running update by the following procedure:

1. Train Neural Network to find $\hat{w}$
2. With $\hat{w}$ as initialization, retrain the model k times with cyclic learning rate to find $\hat{w}_i$
3. Compute $\hat{w}_{swa} = \sum_{i=1}^{T} \frac{1}{T}\hat{w}_i$, $\hat{\Sigma}_w = Diag(\sum_{i=1}^{T} \frac{1}{n}[E[\hat{w}_i^2] - \hat{w}_{swa}^2])$
4. Sample $w \sim N(\hat{w}_{swa}, \hat{\Sigma}_w)$

Even though this estimate has been an naive version with the diagonal covariance structure to indicate independence, a more general low rank outer product estimate of the covariance term is also given but require much more storage complexity in **DNN**. The confidence band is also compared to the one fitted by variational inference in Fig 7, which suggests the **SWAG** contains a more accurate uncertainty interval compared to the variational inference:

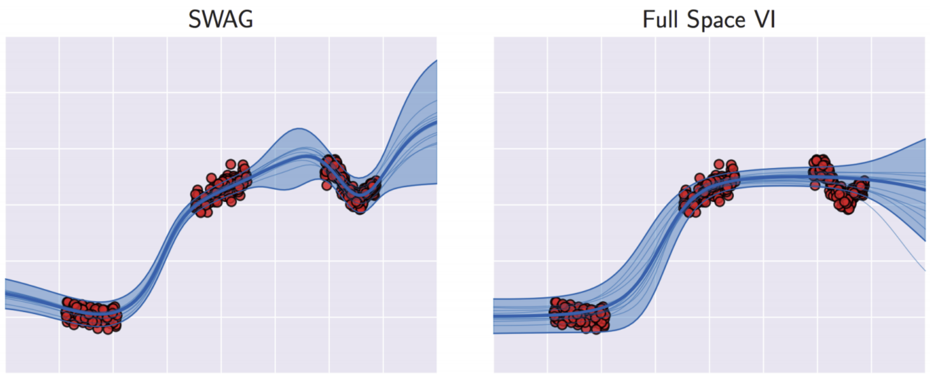

Figure 4: **SWAG Confidence Interval** *[Maddox, 2019]*

3 CIFAR10 Example

Modern software has provided general framework of neural network for implementation. The implementations also come with efficient automatic gradient computation, with parallelization, and with the support of GPU acceleration. Here I adopt pytorch network framework and implement a 3 layers' classifier model on [CIFAR10] dataset. With each class containing 5,000 images the training set contains 10 classes of images('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck'). The testing set contains 1,000 images of each class. The input to my neural network is thus $32 * 32 * 3 = 3072$ due to the RGB three dimensional color scale.

Since the whole course project is about optimization, I didn't use any of the built-in pytorch optimization package. Specifically, The model has 256, 128 and 10 neurons within each of the layer. I trained the model with negative log likelihood loss and with Adam constant learning rate α being tuned by randomly sampling α on a log scale. I have found $\alpha = 0.001$ gives decent result with the optimization loss function given as follow:

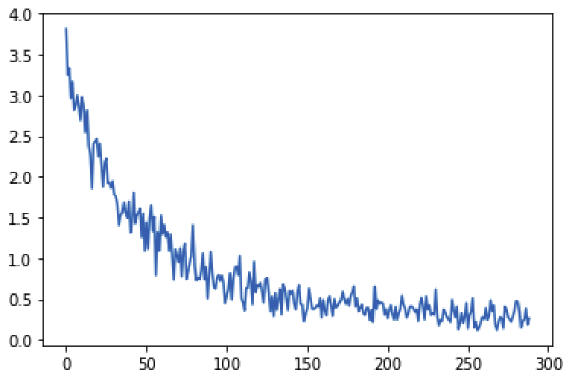

Figure 5: **In sample Training Loss along iteration**

I also implemented the **SWA** with cyclic learning rate (five average decays from 0.1 to 0.01 within 30 steps).The three layers weight distribution are plotted in Fig 6. The weight distribution indicates a minor change in terms of the SWA neural network and the original network, which suggests the original Adam SGD weight $\hat{w}$ is already within the "Flat Minimum" region. Indeed, I found the both of the model with similar classification accuracy score of 49.84% and 49.7% on the testing dateset. Note that the classifier could be improved by using a more complicated neural network structure (eg: CNN), but here I stay with this simple multilayer perceptron model because my propose is to explore the effectiveness of the Adam, SWA and SWAG.

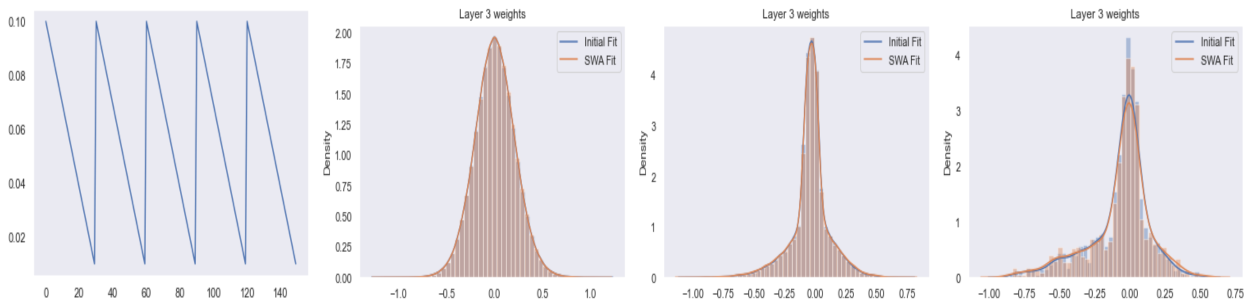

Figure 6: **SWA Result**

During the iteration of SWA, I also computed $\hat{w}_{swa}$ and $\hat{\Sigma}_w$ to prepare for the SWAG model. Then given a specific testing data batch X, I can easily sample S times the neural network weights with $w_s \sim N(\hat{w}_{swa}, \hat{\Sigma}_w)$ and conduct feed forward operation to compute $\hat{P}(Y|X) \approx \frac{1}{S}\sum_{s=1}^{S} p(y|x, w_s)p(w_s)$. Moreover, given sample $w_s, s = 1...S$, I also computed the empirical 95% confidence interval of the probability $\hat{P}(Y|X)$ with one batch of 9 testing data as an example to examine the confidence of the prediction (See Fig 7).

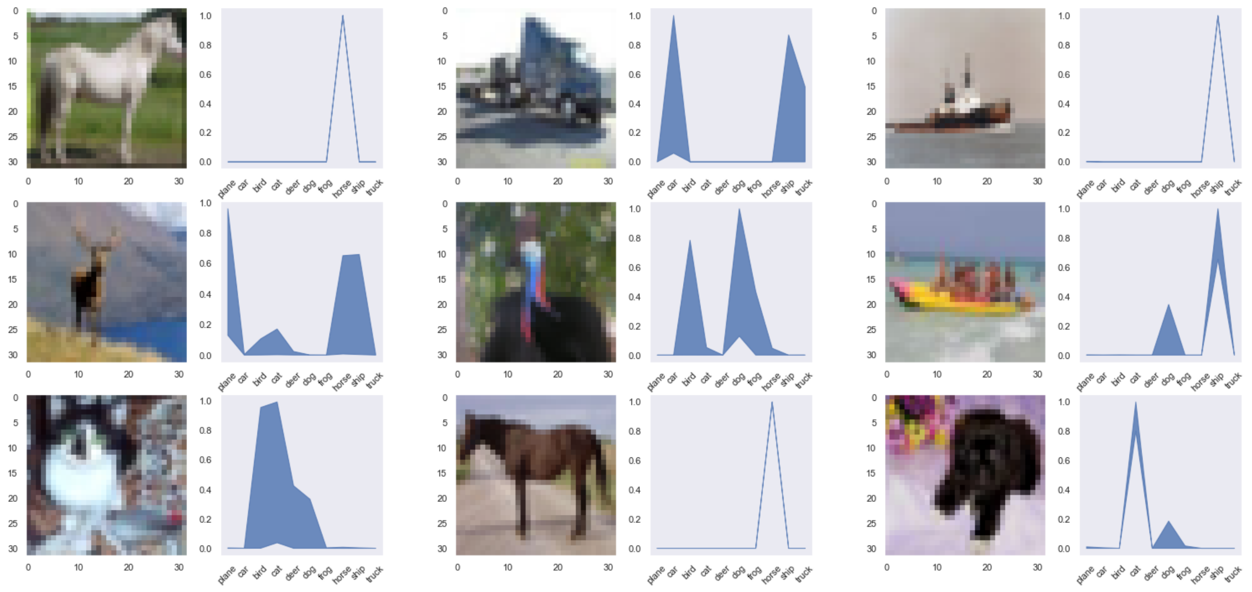

Figure 7: **SWAG Confidence Interval for CIFAR10**

From the confidence interval plot, we could conclude that my neural network can distinguish the horse, cat and ship from the others with high confidence but it seems to be having some troubles in identifying birds, deer and truck.

References

Dmitrii Podoprikhin Timur Garipov, Pavel Izmailov and Andrew Gordon Wilson Dmitry Vetrov. Averaging weights leads to wider optima and better generalization. *Uncertainty in Artificial Intelligence (UAI)*, 2018a.

Garipov Timur Izmailov Pavel Vetrov Dmitry Wilson Andrew Gordon Maddox, Wesley. A simple baseline for bayesian uncertainty in deep learning. *Conference on Neural Information Processing Systems*, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning, PMLR*, 2015.

Maurizio Filippone Simone Rossi, Pietro Michiardi. Good initializations of variational bayes for deep models. *36th International Conference on Machine Learning*, 2018.

Hazan Elad Duchi, John and Yoram. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 2011.

Geoffrey Tieleman, T. Hinton. Neural networks for machine learning. *Technical report*, 2012.

Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

Jorge Nocedal Mikhail Smelyanskiy Nitish Shirish Keskar, Dheevatsa Mudigere and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *International Conference on Learning Representations*, 2017.

Cengiz Pehlevan Blake Bordelon, Abdulkadir Canatar. Spectrum dependent learning curves in kernel regression and wide neural networks. *Journal of Machine Learning Research*, 2020.

Matthew D. Hoffman David M. Blei(2017) Stephan M, t. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 2017.

Jurgen Schmidhuber Fred Cummins Felix A. Gers. Learning to forget: Continual prediction with lstm. *Proc. of the 9th Int. Conf. on Artificial Neural Networks*, 1999.

AlexGravesa Jürgen Schmidhuberab. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 2005.

J. S. Denker D. Henderson R. E. Howard W. Hubbard Y. Le Cun, B. Boser and L. D. Jackel. Handwritten digit recognition with a back-propagation network. *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 1999.

Andrew Ng. Cs229 lecture notes http://cs229.stanford.edu/notes/cs229-notes5.pdf. *Stanford*, 2012.

Yoshua Bengio Razvan Pascanu, Tomas Mikolov. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on International Conference on Machine Learning*, 2013.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

Sara Martino Rue, Havard and Nicolas Chopin. Approximate bayesian inference for latent gaussian models by using integrated nested laplace approximations. *Journal of the Royal Statistical Society, Series B*, 2009.

Sepp Hochreiter Jürgen Schmidhuber. Flat minima. *Neural Computation*, 1997.

Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of statistics*, 1983.

Dmitrii Podoprikhin Timur Garipov, Pavel Izmailov and Andrew Gordon Wilson Dmitry Vetrov. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Conference on Neural Information Processing Systems*, 2018b.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 1992.