BAYESIAN MACHINE LEARNING

Samuel Guilluy
Marwan Wehaiba El Khazen

March 11, 2020

ABSTRACT

In this brief review of [1], we will describe the need for representing uncertainty in deep learning, the challenges it poses, and how the authors set out to resolve it by proposing their general purpose approach: SWAG. We will also consider the theoretical work of [2]that SWAG relies on, as well as the underlying assumptions and whether they are met for DNNs. SWAG is a Gaussian extension of Stochastic Weight Averaging (SWA) that can obtain efficient Bayesian model averaging, as well as high quality uncertainty estimates and calibration in deep learning. The authors have made available their implementation of SWAG on image data sets, so we will expose our own attempts at implementing it on text data sets, and the obstacles we faced.

Keywords Stochastic Weight Averaging Gaussian · Bayesian Machine Learning

1 Introduction

DNNs are powerful models that perform remarkably well across a range of prediction tasks, ranging from language translation to image classification. They often contain hundreds of thousands or even millions of parameters that can be tuned to learn complicated patterns from all sorts of data. In a supervised learning setting, the aim is to learn the distribution $p\left(y|\theta, X\right)$; we search for model parameters θ, say, in $\mathbb{R}^d$, which, given our data X, can be used to discriminate between (or predict) the observations in our target variable y. For any θ, we can calculate how well our model represents the data by using an appropriate loss metric L, such as the commonly used mean-squared error (MSE) in the case of a regression model of negative log likelihood for classification tasks. If we plot L against θ, we can visualize how the loss magnitude varies across parameter space, as in the figure1, provided by the same team of authors in another article:

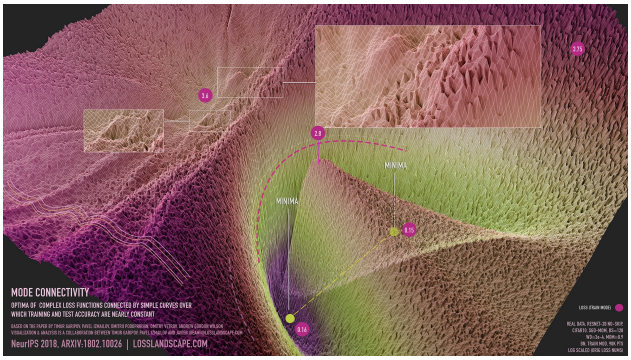

Figure 1: A visualization of the loss surface for RESNET-20 for the CIFAR10 dataset. Credit: Timur Garipov and Colleagues.

This rich illustration zooms into a very frequent and very difficult characteristic of the loss surface: its high non-convexity in real life data. As it happens, this is the loss surface of the RESNET-20 DNN trained on the CIFAR10

dataset. Each θ along the (x, y) axes basically represents a new model, with its error tracked by the height of L along the z-axis; Ideally, we want to pick θ with the smallest value of L. It is therefore an optimization problem.

2 Origin of the model

Loss surfaces in deep learning are often very complex and their geometric properties are not well understood, as soon as we deal with real data. To optimize the loss function, deep neural networks are typically trained with some Stochastic Gradient Descent (SGD) variant, usually with a decaying learning rate, until convergence[3]. But in that framework, SGD will find points on the periphery of a set of good weights. However, by running SGD with a cyclical or high constant learning rate, one can go through the whole surface of this set of points, and by averaging, find a more centred solution in a flatter region of the training loss. In a previous paper,[3], (almost) the same team proposed to follow such a trajectory of weights traversed by SGD, leading to new geometric insights: dubbing this method Stochastic Weight Averaging (SWA), the intuition was that it would lead to better results than standard training. They discovered that SWA indeed improved training of many state-of-the-art deep neural networks, with negligible extra computational cost. On the other hand, [2] had previously shown that under certain assumptions, running SGD with a constant learning rate is equivalent to sampling from a Gaussian distribution centered at the minimum of the loss, and the covariance of this Gaussian is controlled by the learning rate.

Following this proof from [2], and even though the necessary assumptions do not exactly hold for DNNs, the authors then tried to interpret points proposed by SGD as being constrained to the surface of a sphere, since they should come from a multivariate Gaussian distribution. SWA effectively allows to go inside the sphere to find higher density solutions. In a procedure called Fast Geometric Ensembling (FGE), [4] showed that using a cyclical learning rate it is possible to gather models that are spatially close to each other but produce diverse predictions. They used the gathered models to train ensembles with no computational overhead compared to training a single DNN model. In their article, they show that the optima of complex loss functions are actually connected by simple curves over which training and test accuracy are nearly constant, so they sought to discover these high-accuracy pathways between modes. And so, inspired by following the trajectories of FGE (see figure 2), and motivated by the theoretical analysis of the stationary distribution of SGD iterates, the SWA was born. The more recent extension, the Gaussian SWA, or SWAG, proposed by the same team, is the main subject of this review. Its purpose is to further exploit the Gaussian nature of the stationary distribution, by using first and second moments to provide uncertainty estimates along with the more centered/higher density solution offered by SWA.

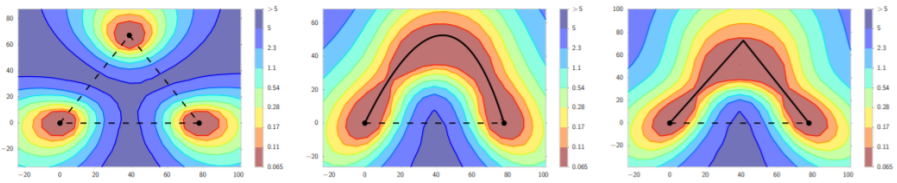

Figure 2: The L2-regularized cross-entropy train loss surface of a ResNet-164 on CIFAR-100, as a function of network weights in a two-dimensional subspace. Left: Three optima for independently trained networks. Middle and Right: A quadratic Bezier curve, and a polygonal chain with one bend, connecting the lower two optima on the left panel along a path of near-constant loss. Notice that in each panel a direct linear path between each mode would incur high loss.

2.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is like a (local) GPS, or a compass, leading us from an initial position down the path of greatest descent until we (ideally) get stuck in the viscinity of one of the (possibly numerous) local minima. An improved version (SGD with momentum) integrates directions of past steps in the decision-making, yielding a smarter and smoother orientation along the surface to be optimized. However, SGD has several limitations: First, as mentioned above, it has been argued that conventional SGD only converges to the boundary of a set of high performing networks.[3] So then small-sample variations can lead to loss surface shifts between training and test datasets that push θ away from this boundary, leading to poor generalization. Second, SGD is often used with a decaying learning rate in order to converge on an optimal θ. While this speeds up inference, it does not provide much coverage to the full space of possible solutions. And third, We neglect the uncertainty associated with θ when we make predictions, leading to overconfident estimates. The authors aim to address these problems in a way that does not demand too much additional computational overhead.

As a simple example, consider one of the proposed SGD, with a constant learning rate (constant SGD). Constant SGD first approaches to an optimum of the objective function and then bounces around its vicinity. (In contrast, traditional SGD would over-confidently converge to the optimum by decreasing the learning rate, with no measure of uncertainty)[2]

Standard training of deep neural networks (DNNs) proceeds by applying stochastic gradient descent on the model weights θ with the following update rule:

$$\Delta\theta_t = -\eta_t \left(\frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \log p(y_i|f_\theta(x_i)) - \frac{\nabla_\theta \log p(\theta)}{N} \right),$$

where the learning rate is η, the ith input and label are $\{x_i, y_i\}$, the size of the whole training set is N, the size of the batch is B, and the DNN, f, has weight parameters θ. The loss function is a negative log likelihood $-\sum_i \log p(y_i|f_\theta(x_i))$, combined with a regularizer $\log p(\theta)$. This type of maximum likelihood training cannot represent uncertainty in the predictions or parameters θ. We note that the previously mentioned SGD with momentum (or Nesterov's Accelerated Gradient) would look like this:

$$\Delta\theta_t = V_t$$
$$V_t = \beta V_{t-1} - \eta_t \left(\frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \log p(y_i|f_\theta(x_i)) - \frac{\nabla_\theta \log p(\theta)}{N} \right),$$

for a tunable value of β, and V_t initialized at 0. This provides a better orientation for the gradient descent, since it has (decaying) memory of the previous necessary directions.[5] A common value of β (and the one used by the authors) is 0.9.

2.2 Stochastic Weight Averaging (SWA)

SWA will be key to solve our first two problems. SWA is a simple procedure that improves generalization in deep learning over Stochastic Gradient Descent (SGD) with no additional cost. It has a wide range of applications and features from computer vision to semi-supervised learning, deep-reinforcement learning, and of course, is the basis for SWAG, which can obtain efficient Bayesian model averaging, as well as high quality uncertainty estimates and calibration in deep learning.

In short, SWA performs an equal average of the weights traversed by SGD with a modified learning rate schedule (see the left panel of Figure 3). SWA solutions end up in the center of a wide flat region of loss, while SGD tends to converge to the boundary of the low-loss region, making it susceptible to the shift between train and test error surfaces (see the middle and right panels of Figure 3). We stress the difference between standard Averaged SGD, which has been used for a long time in convex optimization to improve rates of convergence, and the rather novel concept of equal averaging of SGD iterates with a modified cyclical or high constant learning rate, which SWA explores, and exploits the flatness of training objectives specific to deep learning for improved generalization[4].

SWA can be combined with any optimization procedure in the same way that it can be combined with SGD. It is not tied to one optimizer and we have taken advantage of that fact for our own implementation.

There are two important ingredients that make SWA work. First, SWA uses a modified learning rate schedule so that SGD continues to explore the set of high-performing networks (each one corresponding to a distinct parameter θ) instead of simply converging to a single solution. For example, we can use the standard decaying learning rate strategy for the first 75% of training time, and then set the learning rate to a reasonably high constant value for the remaining 25% of the time (see the Figure 2 below). The second ingredient is to average the weights of the networks traversed by SGD. For example, we can maintain a running average of the weights obtained in the end of every epoch within the last 25% of training time (see Figure 4).

SGD converges to a solution within a wide flat region of loss. The weight space is extremely high-dimensional, and most of the volume of the flat region is concentrated near the boundary, so SGD solutions will always be found near the boundary of the flat region of the loss. SWA on the other hand averages multiple SGD solutions, which allows it to move towards the center of the flat region.

We expect solutions that are centered in the flat region of the loss to generalize better than those near the boundary. Indeed, train and test error surfaces are not perfectly aligned in the weight space. Solutions that are centered in the flat region are not as susceptible to the shifts between train and test error surfaces as those near the boundary. In figure 3

below we show the train loss and test error surfaces along the direction connecting the SWA and SGD solutions. As you can see, while SWA solution has a higher train loss compared to the SGD solution, it is centered in the region of low loss, and has a substantially better test error, making it more robust.

We note the name SWA has two meanings: on the one hand, it is an average of SGD weights. On the other, with a cyclical or constant learning rate, SGD proposals are approximately sampling from the loss surface of the DNN, leading to stochastic weights.[3]

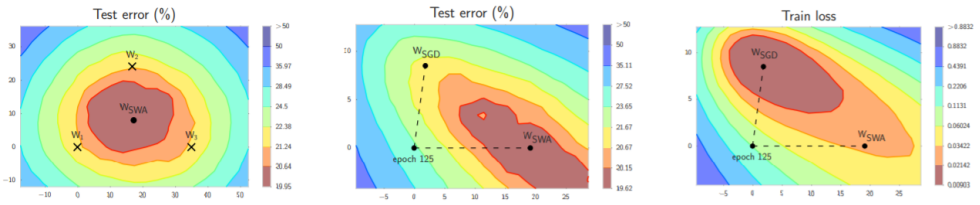

Figure 3: Illustrations of SWA and SGD with a Preactivation ResNet-164 on CIFAR-100. Left: test error surface for three FGE samples and the corresponding SWA solution (averaging in weight space). Middle and Right: test error and train loss surfaces showing the weights proposed by SGD (at convergence) and SWA, starting from the same initialization of SGD after 125 training epochs.

2.3 Bayesian Neural Networks

Bayesian Neural Networks use probability distributions of weights instead of having deterministic weights. Each forward pass will sample different weights θ, and therefore have different outputs. One can see Bayesian Neural Networks as having not a single model, but an ensemble of models, which yields different predictors for each sample. Given that the same input can have different outputs after the forward pass, one can see how certain a given model is by looking at the variance of the different predictions, and if the model is uncertain one can expect the variance to be high.[6]

2.4 The four assumptions

In order for the proof in [2] to work, four assumptions are needed: First, the gradient noise at each point θ should be $N(0, C)$. Second, the value of the covariance C should be independant of θ. Third, the learning rates should be small enough to approximately describe the dynamics of SGD with a stochastic differential equation. Fourth, in the stationary distribution, the loss should be quadratic, and the Hessian matrix around the optimum should be positive definite. In their paper, the authors concede that the second and fourth assumptions are not met. Indeed, for assumption 2, they find that the optimal learning rate predicted by the theory is often 10 times larger than what they had to use in practice. As for the assumption 4, they found negative eigenvalues in the Hessian matrix at the end of some trainings.

3 Presentation of Gaussian Stochastic Weight Averaging (SWAG)

Similarly to SWA, which maintains a running average of SGD iterates, SWAG estimates the first and second moments of the iterates to construct a Gaussian distribution over weights. SWAG distribution approximates the shape of the true posterior by fitting it to a high-dimensional Gaussian distribution and by doing so, is able, to a certain extent, to capture the highly sought-after geometry of the loss function. This enables it to solve the third problem SGD presented, by providing crucial uncertainty estimates for DNNs.

3.1 Theorical presentation

The SWA in SWAG stands for Stochastic Weight Averaging, a method that, as we have described, specifically tackles the first two drawbacks mentioned in section 2.1. The idea is to start from a pre-trained solution θ_{pre}, and then push the learning rate up to explore the local geometry using SGD. At each step i we move to a new position in weight space θ_i at a constant or cyclical learning rate. The constant learning rate gives rise to a new solution which we successively average with our pre-trained solution to give the SWA solution θ_{SWA}. For the cyclical learning rate, the solver "jumps" out of a local minimum and converges to another solution nearby; the converged solution at the end of the learning cycle is averaged with the trained solution. After T epochs, the SWA solution is given by:

$$\theta_{SWA} = \frac{1}{T}\sum_{i=1}^{T}\theta_i$$

The authors show that this new solution is more likely to be centered within a broad set of high performing networks, and highlight improved prediction accuracy across many residual and image classification networks.

Maddox et al. then extend SWA to SWAG by approximating the error estimates using a Gaussian model of the form $N(\theta_{SWA}, \Sigma)$. Σ is a covariance matrix that tracks the variation along the local geometry, and is updated using each new point θ_i along the SWA algorithm's path. Unfortunately, computing Σ is expensive if we include every iteration i; DNNs typically contain millions of parameters. As a speed-hack, the authors use the full set of iterations to update only the diagonal elements in Σ for their SWAG-diagonal algorithm, and the off-diagonals for the general SWAG algorithm are computed using the last K terms along the SGD iterates. Then they can generate an entire distribution given by $N(\theta_{SWA}, \Sigma)$, that characterizes the geometry around θ_{SWA}. This lets them directly model the uncertainty into any given prediction using Bayesian model averaging. In principle, this would involve integrating θ out of $p(y|\theta, X)$ to yield $p(y|X)$. In practice, this is computationally intractable, so they settle for an approximation by Monte Carlo sampling: sampling θ_k $N(\theta_{SWA}, \Sigma)$ K times and averaging $\frac{1}{K}\Sigma p(y|\theta_k, X)$.

In their experiments, the authors showed that SWAG performs as well as or better than frequently used alternatives like MC dropout, KFAC Laplace, and temperature scaling on uncertainty quantification, out-of-distribution detection, calibration and transfer learning in computer vision tasks.

3.2 SWAG-diagonal

To create uncertainty estimates around the SWA average of the SGD iterates, the authors first consider a diagonal covariance matrix format. That requires them to additionally calculate the average of the second moments for the weights and use them to build a diagonal covariance matrix:

$$\overline{\theta^2} = \frac{1}{T}\sum_{i=1}^{T}\theta_i^2$$

$$\Sigma_{\text{diag}} = \texttt{diag}(\overline{\theta^2} - \theta_{SWA}^2)$$

The resulting approximate posterior distribution is then $N(\theta_{\text{SWA}}, \Sigma_{\text{diag}})$.

3.3 SWAG: Low rank plus diagonal structure

The natural extension to diagonal SWAG is to then consider the full covariance matrix of the iterates. Unfortunately, it is prohibitively expensive in computational cost, and so the authors settle for a low-rank matrix and compute an unbiased empirical covariance matrix based on the last T iterates:

$$\Sigma = \frac{1}{T-1}\sum_{i=1}^{T}(\theta_i - \theta_{\text{SWA}})(\theta_i - \theta_{\text{SWA}})^\top$$

However, as seen in section 3.1, $\theta_{textSWA}$ is based on the T epochs, and so ones does not have access to it during training. The solution chosen by the authors is to approximate the sample covariance:

$$\Sigma \approx \frac{1}{T-1}\sum_{i=1}^{T}(\theta_i - \bar{\theta}_i)(\theta_i - \bar{\theta}_i)^\top = \frac{1}{T-1}DD^\top,$$

where D is the "deviation matrix" comprised of columns $D_i = (\theta_i - \bar{\theta}_i)$, and $\bar{\theta}_i$ is the running estimate of the parameters' mean obtained from the first i samples, rather than the complete SWA estimate.

Finally, a last hyperparameter K is introduced, whereby only the last K iterates will be considered ($K \leq T$), thus building the matrix $\widehat{D}$ with columns equal to D_i for $i = T-K+1, \ldots, T$. The corresponding low rank matrix is then:

$$\Sigma_{\text{low-rank}} = \frac{1}{K-1}.\widehat{D}\widehat{D}^\top$$

To obtain a full-rank matrix and use it as their covariance, the authors then take the mean of Σ_{diag} and $\Sigma_{\text{low-rank}}$. The resulting approximate posterior distribution is then $N(\theta_{\text{SWA}}, \frac{1}{2}\cdot(\Sigma_{\text{diag}} + \Sigma_{\text{low-rank}}.))$ They discuss the choice of the constant in the appendix, and claim it was optimal in their experiments. They propose an explanation for the value 0.5: the variance of the weights figures in both matrices, which might explain that averaging works best.

We have summarized below in algorithm 1 the concept used to train SWAG as presented in the paper:

Algorithm 1 Training SWAG

θ_{pre}: pretrained weights; η: learning rate; T: number of steps; c: moment update frequency; K: required rank; S: number of samples

$\overline{\theta} \leftarrow \theta_0,\ \overline{\theta^2} \leftarrow \theta_0^2$ {Initialize moments}
for $i \leftarrow 1, 2, ..., T$ **do**
  $\theta_i \leftarrow \texttt{SGD}(\theta_{i-1})$ {Perform SGD update}
  if `update_time` $=$ `True` **then**
    $n \leftarrow i/c$ {Number of models}
    $\overline{\theta} \leftarrow \dfrac{n\overline{\theta} + \theta_i}{n+1},\ \overline{\theta^2} \leftarrow \dfrac{n\overline{\theta^2} + \theta_i^2}{n+1}$ {Update moments}
    if `nbr_stored_param` $= K$ **then**
      `forget_first_param`
      `store_new_param`$(\theta_i - \overline{\theta})$ {Store deviation}
return $\theta_{\text{SWA}} = \overline{\theta},\ \Sigma_{\text{diag}} = \overline{\theta^2} - \overline{\theta}^2,$ `all_params`

To sample from SWAG, they simply set:

$$\widetilde{\theta} = \theta_{\text{SWA}} + \frac{1}{\sqrt{2}} \cdot \Sigma_{diag}^{\frac{1}{2}} z_1 + \frac{1}{\sqrt{2(K-1)}} \widehat{D} z_2, \quad \text{where } z_1 \sim \mathcal{N}(0, I_d),\ z_2 \sim \mathcal{N}(0, I_K)$$

Proof. Clearly, $\widetilde{\theta}$ is Gaussian as a linear combination of independently Gaussian vectors z_1 and z_2. The expectation of $\widetilde{\theta}$ is trivially θ_{SWA}, since z_1 and z_2 are centered. We now compute the variance, keeping in mind that $\text{Var}(z_1) = I_d$ and $\text{Var}(z_2) = I_K$:

$$\text{Var}(\widetilde{\theta}) = 0 + \frac{1}{2} \cdot \Sigma_{\text{diag}}^{\frac{1}{2}} \cdot \text{Var}(z_1) \cdot \Sigma_{\text{diag}}^{\frac{1}{2}\top} + \frac{1}{2(K-1)} \cdot \widehat{D} \cdot \text{Var}(z_2) \cdot \widehat{D}^\top = \frac{1}{2} \cdot (\Sigma_{\text{diag}} + \Sigma_{\text{low-rank}})$$

□

Algorithm 2 Using SWAG as Bayesian Model Averaging

for $i \leftarrow 1, 2, ..., S$ **do**
  Draw $\widetilde{\theta}_i \sim \mathcal{N}\left(\theta_{\text{SWA}}, \frac{1}{2}\Sigma_{\text{diag}} + \frac{\widehat{D}\widehat{D}^\top}{2(K-1)}\right)$
  $p(y^*|\text{Data}) \mathrel{+}= \frac{1}{S} p(y^*|\widetilde{\theta}_i)$
return $p(y^*|\text{Data})$

4 Presentation of the experimental results

We adapt the code of the SWAG Algorithm in order to be used for text dataset. The code is available online at the address : `https://github.com/wjmaddox/swa_gaussian` We needed to re-implement almost all of the functions we used as they were initially implemented to be used with torchvision for Image analysis. We also used a different NN architecture (embedding+fully connected layer) in order to use an embedding of our data.

4.1 Our implementation of SWAG

The difference between these algorithms and the two previous ones (presented above) is that after starting to store the parameters of our model, we periodically update our model by batch normalization. This method enables us to fully appreciate the impact of the SWAG method as a function of the number of training epochs and also to continually improve our SWAG model.

Algorithm 3 Continuous learning of SWAG model

θ_{pre}: pretrained weights; η: learning rate; T: number of steps; c: moment update frequency; K: required rank; S: number of samples

$\overline{\theta} \leftarrow \theta_0, \ \overline{\theta^2} \leftarrow \theta_0^2$ {Initialize moments}
for $i \leftarrow 1, 2, ..., T$ **do**
  $\theta_i \leftarrow \texttt{SGD}(\theta_{i-1})$ {Perform SGD update}
  if `update_time` $=$ `True` **then**
    $n \leftarrow i/c$ {Number of models}
    $\overline{\theta} \leftarrow \frac{n\overline{\theta} + \theta_i}{n+1}, \ \overline{\theta^2} \leftarrow \frac{n\overline{\theta^2} + \theta_i^2}{n+1}$ {Update moments}
    if `nbr_stored_param` $= K$ **then**
      `forget_first_param`
      `store_new_param`$(\theta_i - \overline{\theta})$ {Store deviation}
  if `estimate_time` $=$ `True` **then**
    for $i \leftarrow 1, 2, ..., S$ **do**
      Draw $\widetilde{\theta}_i \sim \mathcal{N}\left(\theta_{\text{SWA}}, \frac{1}{2}\Sigma_{\text{diag}} + \frac{\widehat{D}\widehat{D}^\top}{2(K-1)}\right)$
      $p(y^*|\text{Data}) \mathrel{+}= \frac{1}{S} p(y^*|\widetilde{\theta}_i)$
return $p(y^*|\text{Data})$

4.1.1 About the hyper parameters

We also need to adjust the hyperparameters. Indeed, in the article, for testing SWAG on CIFAR 10 with the VGG 16 model, they train their model on 300 epochs and start to use SWA after 150 epochs. These numbers are far too high for our model as the SGD stops improving after at most 50 epochs on our dataset.

Thus, during the training phase, we need to adjust the following:

- **Initial learning rate** : if it is too high we will have poor results at the beginning and we therefore risk decreasing to a local minimum.
- **Index of the epoch to start SWA** : We need to have a model which already has good performance in order not to add unnecessary noise to our future estimation of the Bayesian parameters. However, we cannot wait until the full convergence of the SGD method, otherwise our Bayesian Model Averaging (BMA) will provide too close results to the SGD and will have the same issue.
- **SWA learning rate** : It is also unclear how to select the learning rate during the SWA training phase. If we have a too small learning rate, we might oscillate around a local minimum (as opposed to a global one), and if we have a too high learning rate, we might not converge at all.

The figure 4 below illustrates the different steps of the SWAG training program. During the first phase (75%), we have a high learning rate which will then decrease to the final learning rate in the second phase.

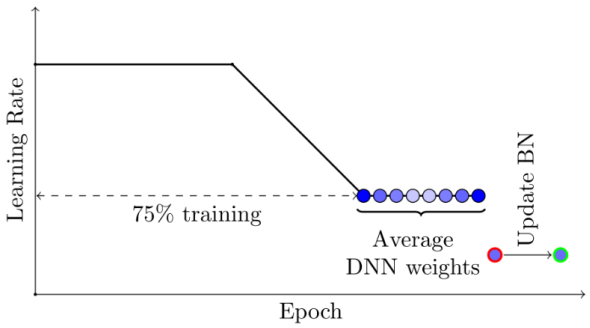

Figure 4: Illustration of the decrease of the learning rate during the training phase of the SWAG method

Then, upon using the BMA model, the parameters are the following:

- Number of models K : In order to reduce the uncertainty of the method, we would like to store an important number of parameters. However, saving many models may lead to memory allocation error as we are using GPU.
- Frequency of Batch updates : we can update the parameters of our Bayesian Machine Learning model only at a given frequency.

4.2 Presentation of the databases

We chose to test our model on two datasets :

- **AG News dataset** The classification task which consists to classify news articles among four categories. This task gives very good results with many methods and thus it is interesting to see how SWAG performs on this task. AG is a collection of more than 1 million news articles. News articles have been gathered from more than 2000 news sources in more than 1 year of activity. `http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html`
- **Yelp Review Dataset** The Yelp dataset is a subset of user reviews which is more difficult to classify. `https://www.yelp.com/dataset`

4.3 Comparison with others optimizers

In order to evaluate the performance of the method, we compare SWAG + SGD with other optimisation algorithms.

In Pytorch, We use the package torch.optim which contains multiple optimizers. As we use a sparse embedding and a linear network, many optimizers in Pytorch do not have the ability to switch from one to another. Therefore, our tests will only consider those who can. We test the following optimizers:

- **SGD + StepLR scheduler** : Classical SGD optimizer with an added Learning Step Scheduler which decays the learning rate of each parameter group by gamma every step size epochs.
- **Adagrad** : Optimizer based on the article [7] which aims to find very predictive but rarely seen features. The algorithm behind it is based on proximal functions to control the gradient steps (convex optimization)
- **SGD + Cosine scheduler** : Stochastic gradient Descent optimizer with the use of a cosine scheduler developed in [8]. The cosine scheduler is a restart technique which aims to improve the rate of convergence in accelerated gradient schemes to deal with ill-conditioned functions. [8]

4.4 Presentation of the experimental results

A global comment on the results we obtained with SWAG+SGD is that the results are pretty stable through the different epochs. Indeed this comes from the fact that we set the index of the epoch to start the SWA algorithm to 5 over 15 epochs which implies our model is already trained with 5 epochs of the SGD optimizer.

4.4.1 Results from the AG News Dataset

As we can see, the accuracy of the different models are pretty close to 1 on the training set (see figure 5) and cap at 91 % on the test set (see the figure 6) . The SWAG method doesn't outperform the other one. We think that the remaining errors on the test came probably from sentences which are ambiguous and should need more pre-processing.

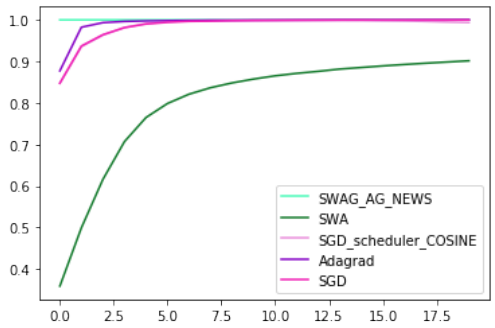

Figure 5: Presentation of the results on the AG News train set

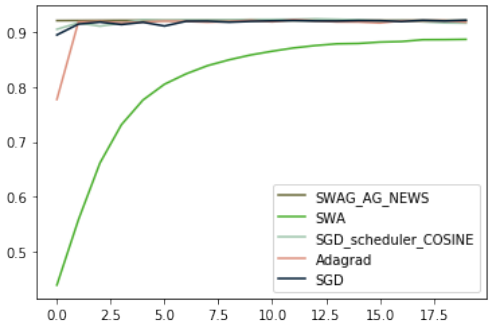

Figure 6: Presentation of the results on the AG News test set

4.4.2 Results from the Yelp Review dataset

As the predictions of the grade given by a person is based on the reading of their reviews, it came as no surprise that the results were lower than on the previous dataset.

The Adagrad optimizer is over-fitting on the training sets (see figure 8, the green curve which decreases, and 7, the highest curve) and thus gives lower results on the test set.

The SWAG, SWA, SGD + Cosine scheduler yield good results both on the test and train datasets. SWA and SWAG oscillate less than SGD + Cosine scheduler as they use a ensemble method. It remains unclear when to use the SWAG algorithm.

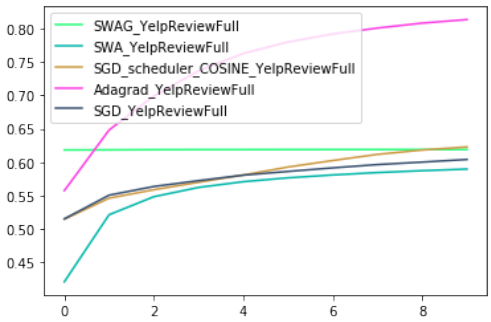

Figure 7: Presentation of the results on the Yelp Review train set

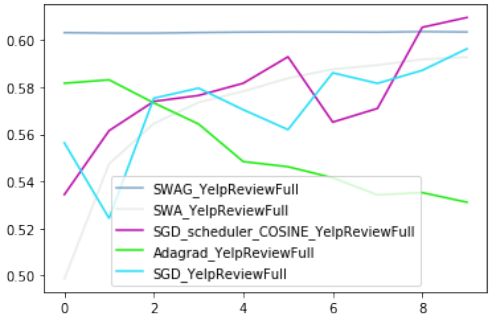

Figure 8: Presentation of the results on the Yelp Review test set

5 Conclusion

SWAG is undoubtedly an important step towards integrating approximate Bayesian inference in deep learning. It is able to understand geometric features such as flatness, and can therefore find optimizers that provide even better generalization, and many other useful features, such as uncertainty representation. SWA and SWAG have demonstratively been shown to offer those features. In our review of [1], we have seen that is is a natural extension of previous work on ensembling and weight averaging. However, we have found it quite difficult to fine-tune, especially with respect to the learning rates of the two phases (see figure 4). We have clearly witnessed the stability of SWAG, as it does not seem to change its predictions over epochs once it stabilizes, however we have not found it to be the most efficient on our datasets, but this may be due to the little fine-tuning we were lead to perform, the nature of some of the datasets (all optimizers were performing quite well quite quickly, we can imagine SWAG being more worthwhile on more complex datasets), or it may be caused by the fact that we tested it using an elementary architecture (embedding + fully connected layer). Finally, for this review, we propose some open questions for which we lack clear answers upon reading this article: - The question of tuning learning rates remains unclear. Which optimizer do we need to use in addition to the SAWG method ? Do we need to use a specific scheduler which perform better with SWAG ? How to we tune it otherwise? - The assumptions of [2] are not met, especially concerning the optimal learning rate and the positive definiteness of the Hessian around the optima. And yet, the empirical results presented in the paper seem to validate the Gaussian posterior approximation used in SWAG in many ways not predicted by the theory. Is there a framework unconstrained by assumptions 2 and 4 for understanding how SWAG captures the geometry of the loss surface so well?

References

[1] Wesley Maddox, Timur Garipov, Pavel Izmailov, Dmitry Vetrov, and Andrew Gordon Wilson. A Simple Baseline for Bayesian Uncertainty in Deep Learning. *arXiv:1902.02476 [cs, stat]*, December 2019. arXiv: 1902.02476.

[2] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. Stochastic Gradient Descent as Approximate Bayesian Inference. *arXiv:1704.04289 [cs, stat]*, January 2018. arXiv: 1704.04289.

[3] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging Weights Leads to Wider Optima and Better Generalization. *arXiv:1803.05407 [cs, stat]*, February 2019. arXiv: 1803.05407.

[4] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. *arXiv:1802.10026 [cs, stat]*, October 2018. arXiv: 1802.10026.

[5] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. page 14.

[6] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. *arXiv:1711.00165 [cs, stat]*, March 2018. arXiv: 1711.00165.

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. page 39.

[8] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv:1608.03983 [cs, math]*, May 2017. arXiv: 1608.03983.