Deep Learning Term-Project Report

HeeWon-Lee

School of Electronics Engineering College, Kyungpook National University

saebuk2000@knu.ac.kr

Ⅰ. Introduction

The dataset used in this study is CIFAR-10, and the model used is the ResNet18 model. We summarize the methods used to improve the performance in a Single Model that were experimented with before the release of the baseline code, as well as the methods used to enhance performance using Ensemble methods after the release of the baseline code.

Ⅱ. Methods for improving performance in Single Models

The following methods were performed before the release of the baseline code.

1. optimizer

The Sharpness-Aware Minimization (SAM)[1] is a powerful and intricate optimization method specifically designed to enhance model generalization. It accomplishes this objective by employing a dual-pronged approach that effectively reduces the loss value while concurrently minimizing the sharpness.

The unique operational mechanism of SAM involves two steps per iteration. During the first step, SAM embarks on the computation of the gradient of the loss in relation to the parameters. It then updates the weights based on this computation, essentially changing the parameters of the model.

Following the update of the weights, the second step of the SAM process is initiated. In this phase, the gradient is recalculated, but this time around the neighborhood of the newly updated parameter values. This effectively minimizes the maximum loss within this specific neighborhood.

Figure 2 schematically illustrates a single SAM parameter update.

SAM's approach of recalculating the gradient around the neighboring relationships of the new parameter values introduces an intriguing feature: low curvature. As represented in Figure 1, this low curvature creates a conducive environment for improved model generalization, setting SAM apart from other optimization methods.

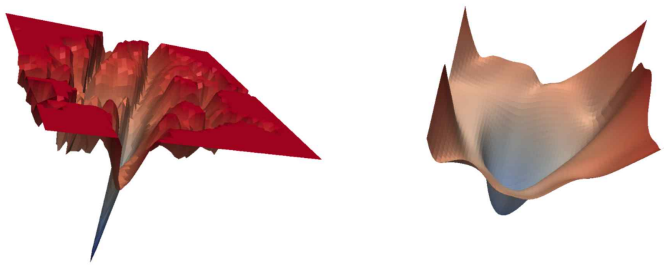

Figure 1. Comparison of Curvature between SGD (left) and SAM (right)

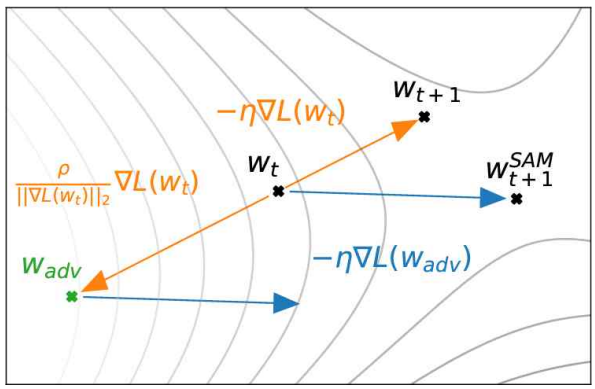

Figure 2. Schematic of the SAM parameter update

However, SAM, like every other method, is not devoid of limitations. It presents the challenge of being sensitive to parameter re-scaling, a critical issue that can affect the consistency of its results. This scale-dependency drawback in SAM creates a unique demand for a method that can effectively address it.

Addressing this limitation is the ASAM, otherwise known as Adaptive SAM [2]. ASAM is a sophisticated optimization method developed specifically to counter the scale-dependency issue in SAM. This is achieved through the implementation of the Adaptive Sharpness Method. ASAM, through this method, manifests a robustness to parameter re-scaling, effectively addressing the identified drawback in SAM. As shown in Figure 3, the ASAM

optimizer converges well with weights, whereas the SAM optimizer does not converge well with weights.

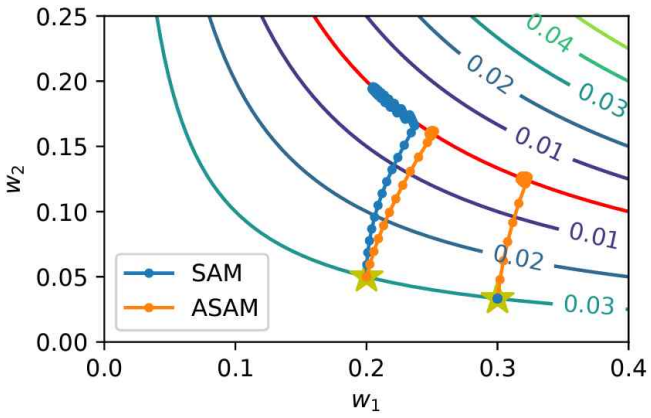

Figure 3. Trajectories of SAM and ASAM.

In the comprehensive Table 1 presented in the ASAM paper, there is a detailed comparison of the performance of Stochastic Gradient Descent (SGD), SAM, and ASAM optimization methods. This comparison provides valuable insights into the relative effectiveness and efficiency of these methods.

In the realm of optimization, the batch size and the learning rate are crucial factors that can influence the performance of models. As indicated in a referenced paper [3], it was explicitly mentioned that the optimal performance when employing SAM or SGD is achieved when the batch size is strategically set to 16, as visually represented in Figure 2.

A similar conclusion was reached in another paper [4], which reported an improved performance when the batch size is maintained at 16 and complemented with a learning rate of 0.0001. Given these findings, we have decided to standardize the batch size at 16 and set the learning rate to 0.0001 in our subsequent operations. This decision is made with a view to optimize model performance and enhance the efficiency of our optimization methods.

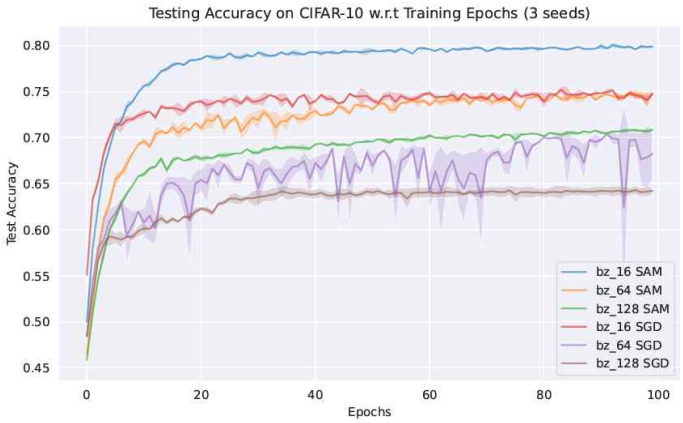

Figure 4. Compare the performance of training CIFAR-10 using SGD and SAM with different batch sizes.

Model	SGD	SAM	ASAM
DenseNet-121	$91.00_{\pm 0.13}$	$92.00_{\pm 0.17}$	$\mathbf{93.33}_{\pm 0.04}$
ResNet-20	$93.18_{\pm 0.21}$	$93.56_{\pm 0.15}$	$\mathbf{93.82}_{\pm 0.17}$
ResNet-56	$94.58_{\pm 0.20}$	$95.18_{\pm 0.15}$	$\mathbf{95.42}_{\pm 0.16}$
VGG19-BN*	$93.87_{\pm 0.09}$	94.60	$\mathbf{95.07}_{\pm 0.05}$
ResNeXt29-32x4d	$95.84_{\pm 0.24}$	$96.34_{\pm 0.30}$	$\mathbf{96.80}_{\pm 0.06}$
WRN-28-2	$95.13_{\pm 0.16}$	$95.74_{\pm 0.08}$	$\mathbf{95.94}_{\pm 0.05}$
WRN-28-10	$96.34_{\pm 0.12}$	$96.98_{\pm 0.04}$	$\mathbf{97.28}_{\pm 0.07}$
PyramidNet-272†	$98.44_{\pm 0.08}$	$98.55_{\pm 0.05}$	$\mathbf{98.68}_{\pm 0.08}$

Table 1. Comparing optimizer performance in CIFAR-10

2. scheduler

In the previously published research paper, specifically denoted as reference [5], the authors provided an insightful observation that, when Stochastic Gradient Descent (SGD) is utilized as the optimization algorithm, the implementation of the Cosine scheduler invariably yields superior performance compared to other scheduling strategies.

Stochastic Gradient Descent, as we know, is a widely accepted optimization algorithm that's frequently used in machine learning and deep learning for training models. This method functions by randomly selecting a small subset of data for computing gradients, which leads to reduced computational requirements and the potential for more diverse gradient calculations. However, one critical aspect that plays a fundamental role in the effectiveness of SGD is the adjustment of the learning rate.

This is where the Cosine LR scheduler comes into play. It modulates the learning rate in a cosine-annealing manner, meaning that it smoothly decreases the learning rate in a cyclical fashion, which can assist SGD in navigating the parameter space more efficiently. This unique ability to dynamically adjust the learning rate throughout the training process allows the model to avoid local minima and makes the SGD algorithm more robust and efficient.

The research paper [5] asserted that this Cosine LR scheduler was the most beneficial choice when utilizing SGD. Consequently, in the context of the present research, where SGD is employed as the fundamental optimizer for Sharpness-Aware Minimization (SAM) and its Adaptive counterpart (ASAM), the Cosine LR scheduler was adopted.

SAM and ASAM are state-of-the-art optimization techniques that emphasize the minimization of sharpness in the loss landscape, aiming to improve generalization performance. In order to maximize their efficiency and effectiveness, it was therefore only logical to integrate the most advantageous learning rate scheduler - the Cosine LR scheduler.

The adoption of the Cosine LR scheduler, therefore, was

not merely a random choice, but a well-informed decision grounded in empirical evidence from previous research. In this manner, the research framework was designed to take full advantage of the proven benefits of the Cosine LR scheduler when used in conjunction with SGD, to optimize the performance of SAM and ASAM.

3. Loss function

I made a strategic choice to apply the Asymmetric Loss function[6]. This decision was primarily driven by my hypothesis that there might exist a considerable mislabeled data or instances of data that pose a significant challenge for humans to categorically differentiate within the CIFAR-10 dataset.

The CIFAR-10, a well-acknowledged dataset in the machine learning field, is replete with intricate data that could potentially blur the boundaries of correct classification. Therefore, it was pivotal for me to select a loss function capable of dealing with such anomalies effectively, and the Asymmetric Loss function emerged as a promising candidate.

Although the Asymmetric Loss function was initially introduced to resolve issues associated with class imbalance and mislabeling in the realm of multi-label classification, its broad applicability allowed me to employ it even within the confines of single-label classification.

Despite this proactive approach and innovation in loss function selection, I regret to acknowledge that I did not conduct extensive experiments to comprehensively compare the performance difference between Asymmetric Loss and the more traditional CrossEntropy Loss. The juxtaposition of these two distinct loss functions could have provided valuable insights into their comparative efficiency and effectiveness when applied to the CIFAR-10 dataset. It remains a missed opportunity in my current study to discern whether Asymmetric Loss actually outperforms CrossEntropy Loss in dealing with the speculated data issues.

4. Data Augmentation And DataLoader

The method of Data Augmentation implemented in our model builds heavily upon the approach described in reference [7]. In addition to this, we incorporated Albumentation, which is a powerful image augmentation library detailed in reference [8]. Its application in our training process is not merely as an auxiliary tool, but it is an integral part of the training regimen, significantly contributing to an increase in the model's robustness, precision, and speed.

Further efforts were made to enhance the training speed by tweaking certain parameters in the DataLoader. Notably, we set the parameter pin_memory to True, which assists in speeding up data transfer between the CPU and GPU. When the pin_memory is set to True, it instructs the Pytorch DataLoader to use pinned (page-locked) memory, which leads to faster data transfer by reducing the time spent on memory allocation and deallocation.

Moreover, we increased the num_workers parameter to 32 in DataLoader. By increasing its value, we can facilitate simultaneous data loading while model training is occurring, leading to a more efficient utilization of computational resources, and consequently, a significant improvement in the training speed.

5. Model EMA

To optimize the effectiveness and efficiency of the model training process, a technique known as exponential moving average (EMA), referenced as [9] in literature, was implemented. This method, also known as EMA, operates on the principle of calculating a kind of moving average of the current weights during training and using this averaged value when proceeding with updates for new weights.

Let's explore the concept of EMA more in depth. In traditional machine learning models, the process of training involves repeatedly adjusting the model's parameters in such a way that the model's error on a given training dataset is minimized. This usually involves some form of gradient descent, where the model's parameters are iteratively adjusted in the direction that reduces the model's error the most.

The Exponential Moving Average technique brings in a slight variation to this process. Instead of using the raw, latest values of the parameters for each update, it computes a running weighted average of the parameters, where more recent values are given higher weightage. This weighted average then serves as the basis for the update. EMA is a special type of moving average that gives more weight to recent data points, making it quicker to respond to changes compared to a simple moving average.

This method has several advantages that result in a more reliable and stable training process. First, it helps in reducing the high frequency noise that can occur in the gradient updates, which can otherwise lead to instability in the training process. Second, EMA can help the model

parameters converge more quickly to the optimal values, as it combines information from the past and present updates.

By leveraging the EMA technique during model training, the variability in model parameters is significantly reduced. This has the advantage of preventing drastic swings in parameter values, which can result in a more consistent and stable training process. This ultimately improves the ability of the model to generalize to unseen data, as it becomes less likely to overfit to the specific quirks of the training data.

In conclusion, the exponential moving average is a powerful tool in machine learning, as it allows for faster and more stable convergence during the training process, ultimately maximizing the generalization capability of the model.

6. Experiment

I applied the same method and conducted the training. The training results are as follows: Train Loss is 0.001918, Validation Loss is 0.07747, Train accuracy is 0.9999, and Validation accuracy is 0.9741. You can see the Accuracy graph in Figure 3.

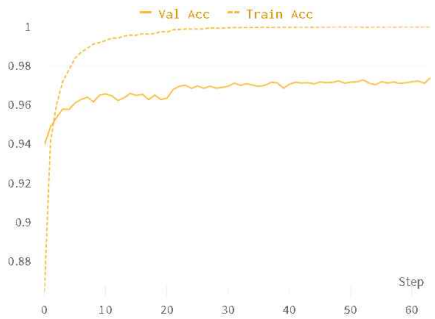

Figure 3. Accuracy graph according to epoch

III. Performance Improvement in Ensemble Models

1. Methods change

When evaluating the performance by training multiple models using the methods from the previous chapter and then ensembling them, the performance did not improve significantly.

Since the performance increased more when applying ASAM compared to not applying it for ensembling, ASAM was not applied. Instead, Adam was used with parameters set according to reference [4].

As the optimizer was changed to Adam, the method from [5] for scheduler could not be applied. Therefore, StepLR from the baseline was used.

Data augmentation was performed using Albumentation for training, but when inferring with the baseline inference code, the performance decreased significantly. Hence, transforms were used for both training and inference.

The loss function remained unchanged, and Model EMA was not used as its performance was higher when not applied.

2. Experiment

I trained the model by changing the random seed from 0 to 10. When inferring with different numbers of models, the result was highest when ensemble with 6 models was used.

During inference, I collected the predicted values for each data and determined the final prediction by selecting the label with the highest frequency. However, this method has a potential problem of making incorrect predictions when there are multiple labels with the same highest frequency. To address this issue, I applied the soft voting method, which helped resolve the problem and achieve even higher performance.

Using the aforementioned methods, the model achieved an accuracy of 97.68%.

IV. Conclusion

My study aimed at improving the performance of a ResNet18 model trained on the CIFAR-10 dataset by implementing various methodologies. Initially, we used optimization techniques, SAM and ASAM, and established that ASAM demonstrated superior performance due to its robustness to parameter re-scaling. Our experimentation further established a batch size of 16 and a learning rate of 0.0001 as the optimal settings when using either SAM or SGD. Furthermore, we utilized the cosine LR scheduler, Asymmetric Loss function, and implemented data augmentation methods to enhance the model's performance. The application of the exponential moving average (EMA) allowed for faster, more stable convergence of the model parameters.

Following these enhancements, the single model achieved a training accuracy of 99.99% and validation accuracy of 97.41%. However, we noted that ensembling multiple models trained using the same methods did not significantly improve performance. Therefore, we adapted our approach for the ensemble model, switching to the Adam optimizer and StepLR scheduler. We continued using data augmentation but replaced Albumentation with transforms for both training and inference to counteract a drop in performance. We also found that Model EMA negatively impacted the ensemble's performance and thus discontinued

its use.

By adjusting the ensemble approach, including changing the random seed from 0 to 10 and incorporating soft voting during inference, we improved the model's accuracy to 97.68%. It is noteworthy that even minute modifications to parameters and methods can significantly enhance model performance. Future research could focus on exploring other optimization methods, loss functions, and ensembling techniques to drive further improvements in performance.

V. Acknowledgments

This paper was written as a result of the Term-project conducted in the 'Deep Learning' lecture at Kyungpook National University Graduate School of Artificial Intelligence.

References

[1] Foret, Pierre, et al. "Sharpness-aware minimization for efficiently improving generalization." arXiv preprint arXiv:2010.01412 (2020).

[2] Kwon, Jungmin, et al. "Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks." International Conference on Machine Learning. PMLR, 2021.

[3] Coldenhoff, Jozef Marus, Chengkun Li, and Yurui Zhu. "Model Generalization: A Sharpness Aware Optimization Perspective." arXiv preprint arXiv:2208.06915 (2022).

[4] Kandel, Ibrahem, and Mauro Castelli. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset." ICT express 6.4 (2020): 312-315.

[5] Moreau, Thomas, et al. "Benchopt: Reproducible, efficient and collaborative optimization benchmarks." Advances in Neural Information Processing Systems 35 (2022): 25404-25421.

[6] Ridnik, Tal, et al. "Asymmetric loss for multi-label classification." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021.

[7] https://www.kaggle.com/code/ayushnitb/cifar10-custom-resnet-cnn-pytorch-97-acc

[8] Buslaev, Alexander, et al. "Albumentations: fast and flexible image augmentations." Information 11.2 (2020): 125.

[9] Cai, Zhaowei, et al. "Exponential moving average normalization for self-supervised and semi-supervised learning." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.