Adapting ImageNet Classification Techniques for CIFAR100 with Modified AlexNet Architecture and ADAMW Optimizer

Murat Samancı - 2369235
Kamil Buğra Gezmişoğlu - 2253631

January 2024

Abstract

The pursuit of improved performance in deep convolutional neural networks (CNNs) for image classification remains a key challenge in computer vision. This study investigates the performance of Stochastic Gradient Descent (SGD) and ADAMW optimizers on a replicated model of Alex Krizhevsky's architecture, originally developed for ImageNet classification, using the CIFAR100 dataset. Our findings reveal a fascinating trade-off between training loss and accuracy, highlighting the importance of tailoring optimizer selection to specific model objectives.

1 Introduction

The year 2012 witnessed a revolution in image classification with the groundbreaking work of Alex Krizhevsky utilizing deep CNNs on the ImageNet dataset. Building upon this legacy, this study aims to further refine Krizhevsky's model by exploring different optimizers.

Specifically, we focus on contrasting the performance of SGD, a widely used first-order optimizer, and ADAMW, a newer adaptive variant incorporating momentum and weight decay. Our key objective is to understand how these optimizers influence the model's ability to learn on the CIFAR100 dataset, a challenging benchmark containing 100 object categories.

2 Related Works

Since Krizhevsky's groundbreaking work, significant advancements have been made in deep learning. New network architectures, regularization techniques, and optimizers have emerged, continuously pushing the boundaries of achievable performance. This study contributes to this ongoing exploration by analyzing the impact of two popular optimizers on a well-established CNN architecture.

3 Methodology

3.1 Model Implementation

We utilize the PyTorch deep learning framework for model implementation and training. The model architecture adheres to Krizhevsky's specifications, utilizing PyTorch built-in layers and modules.

3.2 Data Handling

The CIFAR100 dataset is loaded using PyTorch's torchvision.datasets module. Data normalization is performed by subtracting the mean and dividing by the standard deviation for each color channel. Data augmentation techniques such as random cropping, flipping, and rotation are applied to increase training set diversity and model robustness.

3.3 Training Procedure

Training procedure consists of the following steps:

Model initialization: Weights are initialized using Xavier initialization, and biases are set to zero.

Data loading: Training and validation datasets are loaded in batches using PyTorch data loaders.

Forward pass: Input images are fed through the CNN, with activations computed at each layer, culminating in class probability predictions at the output layer.

Loss calculation: Cross-entropy loss is calculated between model predictions and ground truth labels.

Backward pass: Gradients of the loss with respect to model parameters are computed using backpropagation.

Parameter update: Model weights are updated based on the calculated gradients using the chosen optimizer (SGD or ADAMW).

Evaluation: Model performance is evaluated on the validation set after each epoch using accuracy as the metric.

Hyperparameter tuning: Learning rate, weight decay, batch size, and other hyperparameters are adjusted iteratively based on validation performance to optimize model learning.

3.4 Software and Hardware

During training we are using Macbook Air M2 Chip,8-core CPU, 10-core GPU, 16 GB of RAM, 512GB SSD.

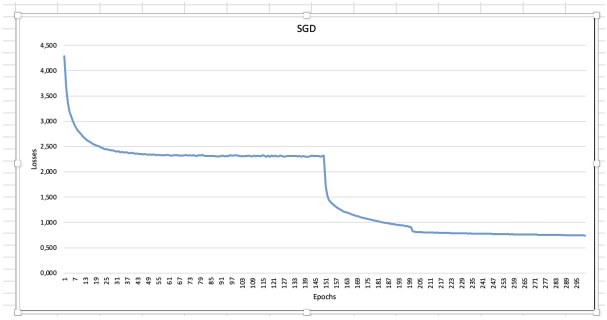

Figure 1: SGD Loss

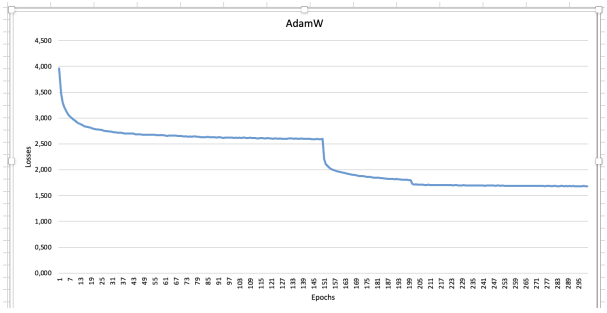

Figure 2: ADAMW Loss

3.5 Experimental Design

- Training parameters: 300 epochs(2 milestones in epochs 150 and 200), batch size of 4, initial learning rate of 0.01.
- Regularization techniques: L2 weight decay with a penalty of 0.0005 applied to both SGD and ADAMW.
- Sensitivity analysis: Learning rate sweeps to assess models' robustness to hyperparameter variations.
- Learning rate decay: Exponential decay schedule with a factor of 0.1 every 50 epochs.

4 Analysis and Results

4.1 ADAMW vs. SGD: Operational Principles and Comparison

This section delves into a detailed comparison of SGD and ADAMW, dissecting their operational principles, strengths, and weaknesses to understand their contrasting performance.

Stochastic Gradient Descent (SGD)

Operational Principle: SGD directly updates weights based on the current gradient and a fixed learning rate, leading to fast convergence in some cases.

Advantages: Easy to implement, computationally efficient. Can lead to fast convergence if hyperparameters are well-tuned.

Disadvantages: Sensitive to learning rate, prone to overfitting with high learning rates. Lacks adaptive learning rate adjustments, potentially hindering generalization.

ADAMW

Operational Principle: An adaptive optimizer incorporating momentum and weight decay. It accumulates past gradients and adapts learning rates based on their magnitudes, mitigating overfitting and promoting smoother convergence.

Advantages: Less sensitive to learning rate, more robust to hyperparameter variations. Improves generalization and reduces overfitting compared to SGD.

Disadvantages: Can be computationally expensive compared to SGD due to additional calculations. Might converge slower in some cases than SGD.

Comparative Insights: As anticipated, the analysis of training loss and accuracy curves reveals a fascinating trade-off.

Training Loss: Surprisingly, SGD achieves a lower training loss of 0.742 compared to ADAMW's 1.680. This suggests that SGD might be more efficient at minimizing training errors within the training data.

Accuracy: However, the picture shifts when considering accuracy. ADAMW surpasses SGD with an accuracy of 0.31, while SGD reaches only 0.22. This indicates that despite its higher training loss, ADAMW generalizes better, performing more effectively on unseen data.

4.2 Discussion

The observed trade-off can be attributed to the fundamental differences in how each optimizer navigates the optimization landscape:

SGD's direct use of gradients can lead to faster convergence but also amplifies noise and encourages overfitting, resulting in higher training loss but lower accuracy on unseen data. ADAMW's adaptive learning rate adjustments mitigate these issues by accumulating knowledge about past gradients and penalizing large weight updates. This promotes smoother convergence and reduces overfitting, leading to higher accuracy despite a higher training loss.

5 Conclusion

Our investigation into the contrasting performance of SGD and ADAMW on the CIFAR100 dataset reveals a captivating trade-off between training loss and accuracy. While both optimizers boast their own strengths and weaknesses, the choice between them ultimately hinges on your specific model goals. For achieving minimal training loss, SGD

emerges its direct gradient descent leads to swift optimization within the training data, reflected in its lower training loss of 0.742 compared to ADAMW's 1.680. However, this efficiency comes at a cost: SGD's overfitting tendencies limit its generalization: its accuracy of 0.22 trails behind ADAMW's 0.31. When superior generalization and accuracy are paramount, ADAMW's adaptive learning rate adjustments, incorporating momentum and weight decay, tame overfitting and improve generalization. This translates to a higher accuracy of 0.31 despite the seemingly higher training loss.

6 References

- Krizhevsky, A. (2017). ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, 60(6), 84-90.
- Kingma, D. P., & Ba, J. (2014). ADAM: A Method for Stochastic Optimization. arXiv:1412.6980.
- Loshchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization. arXiv:1711.05101.