AI6103 Project Report
Effects of Hyperparameters on Network Training and Generalization

Sha Long
G2402907A
C240102@e.ntu.edu.sg

Abstract

In this project, we investigate the effects of hyperparameters on MobileNet's performance using CIFAR-100. Experiments show that a learning rate of 0.05 balances convergence and overfitting, while cosine annealing improves generalization. Weight decay regularization with λ=0.0005 provides stronger regularization compared to λ=0.0001. ReLU outperforms Sigmoid in terms of training speed and accuracy. The best configuration—learning rate of 0.05, cosine annealing, weight decay of 0.0005, and ReLU activation—yields the most accurate results.

1. Introduction

Hyperparameter tuning plays a crucial role in determining the performance of deep neural networks. In this report, we explore how various hyperparameters—learning rate, learning rate schedule, weight decay, and activation function—affect the training and validation performance of the MobileNet model trained on the CIFAR-100 dataset. MobileNet, known for its efficiency and scalability, serves as an ideal candidate for this investigation.

Rather than seeking an optimized configuration, the focus of this study is to examine the influence of different hyperparameter settings on the model's behavior. Through a series of experiments, we analyze the effects of each hyperparameter on training dynamics, such as convergence rates, overfitting tendencies, and generalization capabilities. By reflecting on the results, we aim to provide insights into how these hyperparameters impact the model's ability to learn effectively.

The findings from this exploration offer valuable observations that can inform future decisions when tuning deep learning models, emphasizing the importance of understanding the trade-offs associated with hyperparameter choices.

Learning Rate

In this section, we evaluate the effect of different learning rates on model performance. The models were trained without any learning rate schedule, with weight decay set to 0, and using ReLU as the activation function. We compare the results for learning rates of 0.01, 0.05, and 0.2, and the graphs display both training and validation loss and accuracy.

Learning Rate = 0.01 At a learning rate of 0.01, the model achieved high training accuracy but showed signs of significant overfitting. As seen in Figure 1, validation accuracy plateaued early, and validation loss increased, indicating poor generalization to unseen data.

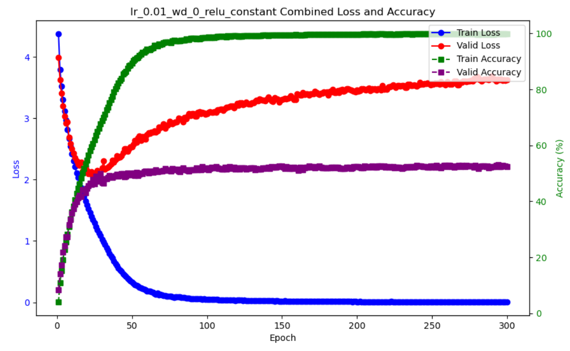

Figure 1: learning rate = 0.01

Learning Rate = 0.05 At a learning rate of 0.05, the model demonstrated improved generalization, as shown in Figure 2. While the validation loss increased over time, the final validation accuracy was slightly higher than that achieved with a learning rate of 0.01, indicating better overall performance.

Learning Rate = 0.2 With a higher learning rate of 0.2, the model converged quickly, as illustrated in Figure 3. However, compared to lower learning rates, the validation loss continued to increase after epoch 200, signaling overfitting. Interestingly, the validation accuracy was higher than at lower learning rates, despite the increasing loss, indicating that the model was still generalizing better on the validation set.

Comparison

From the comparison of different learning rates, it is evident that selecting the appropriate learning rate is critical for

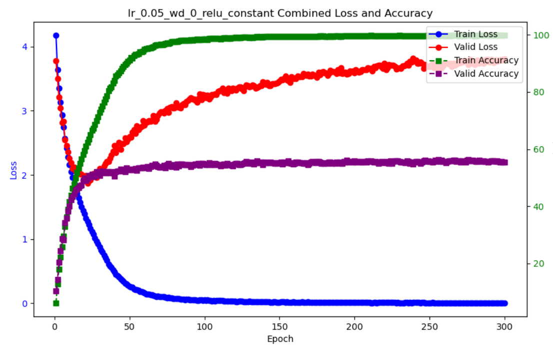

Figure 2: learning rate = 0.05

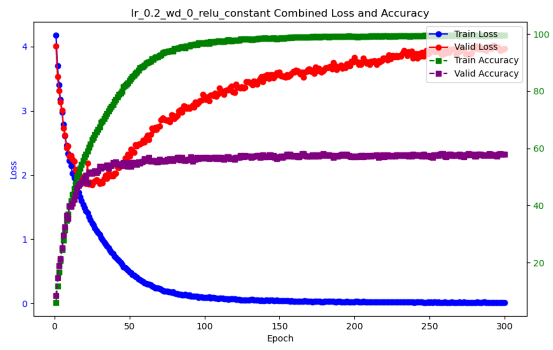

Figure 3: learning rate = 0.2

training. A lower learning rate of 0.01 results in slower convergence and overfitting, while a higher rate of 0.2 leads to rapid overfitting. The learning rate of 0.05 strikes a balance between convergence speed and generalization, showing the most stable results with minimal overfitting. Interestingly, as the learning rate increases, the final validation accuracy also increases slightly, despite the rising validation loss. This could be due to:

- Faster convergence and escaping local minima.
- A more efficient optimization path, as the validation accuracy had not yet fully plateaued after 300 epochs.

Learning Rate Schedule

A learning rate schedule dynamically adjusts the learning rate during training to enhance model performance. In this section, we compare the effects of different learning rate schedules on a model trained with a fixed learning rate of 0.05, no weight decay, and RELU activation.

The following results demonstrate the performance of the model using cosine annealing and constant learning rate schedules.

Cosine Annealing Schedule The cosine annealing schedule smoothly reduces the learning rate as training progresses, resembling a cosine function. As illustrated in Figure 4, the

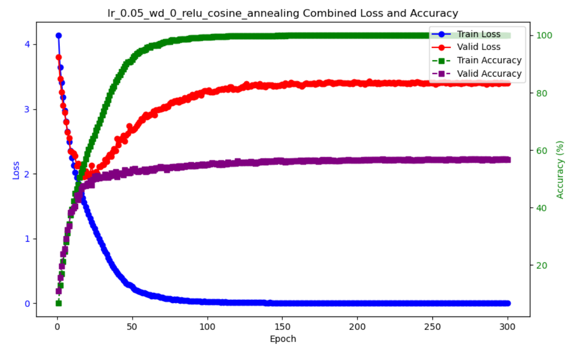

Figure 4: Cosine annealing learning rate schedule

model's training loss declines steadily while the validation loss remains stable, indicating a robust generalization to the validation set. The model achieves a validation accuracy of about 60% and lowers the final validation loss to about 3, with little fluctuation in performance.

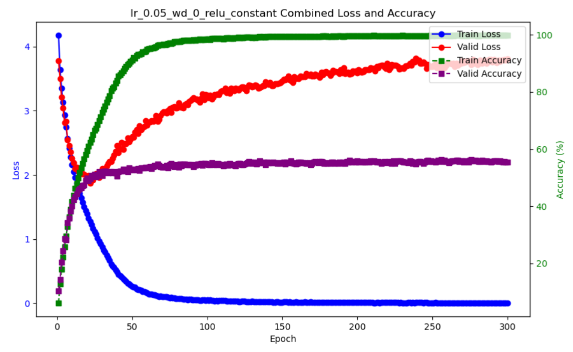

Figure 5: Constant learning rate schedule

Constant Learning Rate Schedule Figure 5 displays the training and validation performance when a constant learning rate of 0.05 is used throughout training. This schedule produces more stable and predictable training loss and validation accuracy, though it lacks the adaptability of the cosine annealing approach. The validation accuracy stabilizes around 60%, similar to the cosine schedule, but the model experiences significantly higher validation loss.

Comparison

Overall, the cosine annealing schedule offers better control over the learning process by smoothly reducing the learning rate, leading to a more controlled training loss without much impact on the validation accuracy. On the other hand, the constant learning rate is simpler but can result in less flexibility, particularly in the later stages of training.

Weight Decay

Weight decay, also known as L_2 regularization, is a technique used to prevent overfitting by penalizing large weights in the model. In this section, we compare the effects of different weight decay values on a model trained with a fixed learning rate of 0.05, no learning rate schedule, and RELU activation.

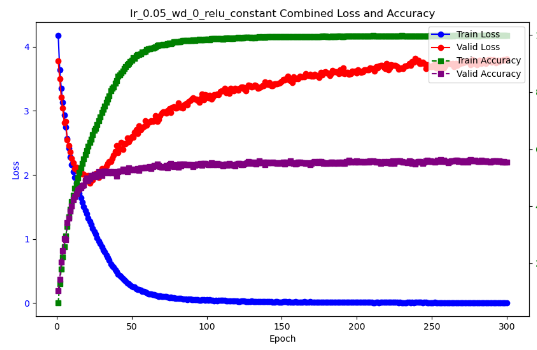

Figure 6: no weight decay

No Weight Decay In the absence of weight decay, the model achieves high training accuracy, but validation accuracy stabilizes around 60%, as shown in Figure 6. The gap between training and validation accuracy is noticeable, indicating a potential for overfitting, as training loss decreases substantially while validation loss remains relatively higher.

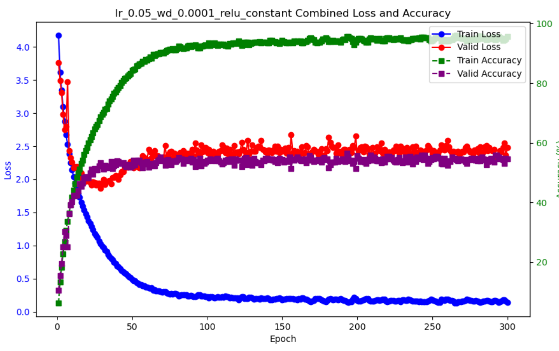

Figure 7: weight decay = 0.0001

Weight Decay = 0.0001 With a small weight decay of 0.0001, as seen in Figure 7, the model shows more balanced performance between training and validation sets. Training loss is slightly higher than without weight decay, but validation loss is reduced. This suggests that the model generalizes better, with less overfitting, though validation accuracy remains around the same level as the no-weight-decay case.

Weight Decay = 0.0005 Increasing the weight decay to 0.0005, as shown in Figure 8, results in even higher training loss but significantly reduced validation loss. This further

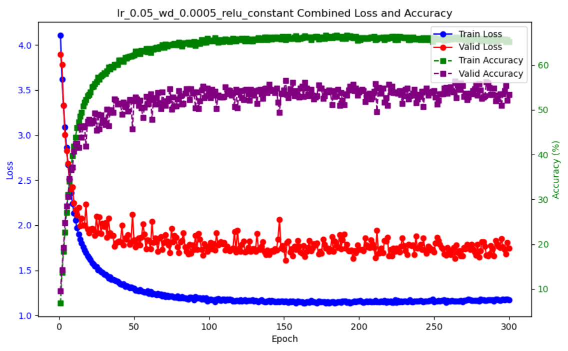

Figure 8: weight decay = 0.0005

improvement in generalization comes at the cost of slower training convergence, but validation accuracy stabilizes similarly to the other cases. The stronger regularization leads to a narrower gap between training and validation performance, indicating better robustness against overfitting.

Comparison

From this comparison, we can conclude that applying weight decay helps mitigate overfitting by reducing the gap between training and validation performance. A small weight decay of 0.0001 strikes a good balance between generalization and training efficiency, while a higher weight decay of 0.0005 further improves generalization but requires more epochs to achieve stable performance. However, even after 300 epochs, the training has not fully stabilized, with both loss and accuracy continuing to exhibit some fluctuations.

Activation Functions

The choice of activation function plays a critical role in model performance, affecting both convergence and accuracy. In this section, we explore the effects of using different activation functions under the same training configuration: a learning rate of 0.05, no weight decay, and no learning rate schedule. Specifically, we compare the performance of ReLU and Sigmoid activation functions.

ReLU Activation The following figure shows the performance of the model using the ReLU activation function. The model demonstrates steady training and validation accuracy, with validation loss decreasing over time but stabilizing at a higher level compared to the Sigmoid activation. ReLU leads to faster convergence, with training accuracy approaching near 100% and validation accuracy stabilizing around 60%. While the ReLU function performs well overall, the validation loss remains higher compared to Sigmoid, suggesting the model may not generalize as effectively under this activation function.

Sigmoid Activation The Sigmoid activation function shows similar performance to ReLU in terms of validation accuracy, but with lower validation loss throughout the training process. This indicates that the model generalizes

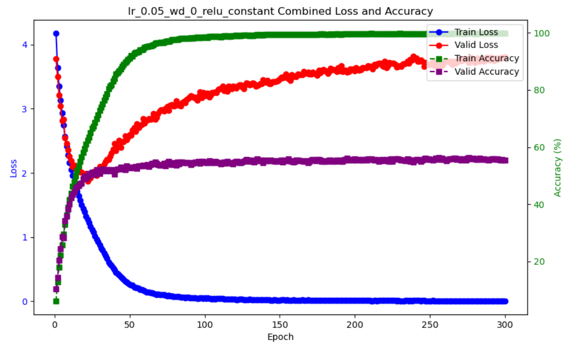

Figure 9: ReLU

slightly better with Sigmoid, despite converging at a similar pace to ReLU. The validation accuracy stabilizes around the same point as ReLU (around 60%), but the lower validation loss suggests that the model fits the unseen data more effectively under Sigmoid.

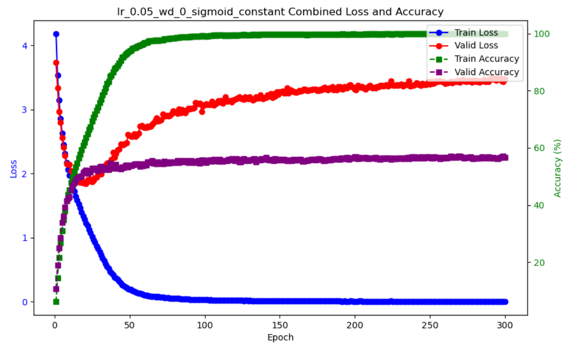

Figure 10: Sigmoid

Comparison

From the results, we observe that both the ReLU and Sigmoid activation functions perform similarly in terms of validation accuracy, with both stabilizing around 60%. However, Sigmoid provides a slight advantage in terms of lower validation loss, indicating that the model generalizes better with Sigmoid activation. ReLU offers faster convergence but results in a higher validation loss, which may indicate less effective generalization to unseen data. Overall, Sigmoid shows better performance in preventing overfitting while maintaining similar validation accuracy.

Conclusion

From the analysis conducted in the previous sections, several important insights about model performance emerge. These insights are derived from systematic comparisons of learning rates, learning rate schedules, weight decay, and activation functions.

Learning Rate

The learning rate plays a vital role in determining the model's performance and generalization capabilities. A low learning rate of 0.01 can cause slow convergence and significant overfitting, whereas a higher learning rate of 0.2 accelerates convergence but leads to overfitting after a certain point. The intermediate learning rate of 0.05 provides the most balanced performance, with a reasonable trade-off between training speed and generalization. While validation loss increases at higher learning rates, the slight improvement in validation accuracy suggests that the model benefits from faster learning and a more effective optimization path. Therefore, a moderate learning rate, such as 0.05, appears to be optimal in balancing training efficiency and model generalization, preventing extreme overfitting while still achieving strong performance.

Learning Rate Schedule

Implementing a learning rate schedule can have a significant impact on model performance, particularly in managing training loss and generalization. The cosine annealing schedule, with its dynamic adjustment of the learning rate, demonstrates better control over the learning process by steadily reducing the rate, leading to lower final validation loss and more stable training. While the constant learning rate approach results in comparable validation accuracy, it struggles with higher validation loss, indicating that the model may not generalize as effectively. Therefore, using a cosine annealing schedule is advantageous for models requiring sustained training over many epochs, as it helps avoid overfitting and maintains performance as the training progresses.

Weight Decay

Weight decay proves to be an effective regularization technique for improving model generalization and mitigating overfitting. As demonstrated, increasing weight decay leads to a narrower gap between training and validation performance, with reduced validation loss as regularization strengthens. A small weight decay value of 0.0001 balances generalization and training efficiency, while a higher weight decay of 0.0005 further enhances generalization at the expense of slower convergence and higher training loss. Ultimately, weight decay helps maintain model robustness by controlling overfitting, though additional epochs may be necessary to achieve more stable performance, especially at higher decay rates.

Activation Functions

Under the specific training configuration of a learning rate of 0.05, no weight decay, and no learning rate schedule, both ReLU and Sigmoid activation functions perform similarly in terms of validation accuracy, stabilizing around 60%. However, Sigmoid demonstrates a slight advantage by achieving lower validation loss, suggesting better generalization and improved model performance on unseen data. While ReLU offers faster convergence and higher training accuracy, its higher validation loss indicates a potential for overfitting. Therefore, within this setup, Sigmoid proves to be more effective in balancing model performance, particularly in preventing overfitting, without sacrificing validation accuracy. This highlights the importance of carefully selecting activation functions depending on the specific requirements of model generalization and training dynamics, especially when no other regularization techniques or learning rate schedules are applied.

Takeaway

This training experience has helped me understand a lot more about how different techniques in deep learning affect model performance. Before, I only knew these things from lecture slides without really knowing why they mattered. Now, I've seen for myself how adjusting things like learning rates and activation functions can change the way a model learns and generalizes. One of the biggest lessons I've learned is how sensitive models can be to small changes—sometimes tiny adjustments can lead to noticeable differences. I also realized how important it is to pay attention to overfitting and balance that with training speed. Beyond that, I learned that the process is very hands-on and requires a lot of experimentation, which isn't something I expected from just reading the theory. This experience has made me feel more confident in understanding how these methods work in practice.

Code Availability

The full source code and implementation details are available on GitHub: `https://github.com/SalonetheGreat/AI6103-Project`.