Performance of ResNet Architecture with Different SGD Optimizers

EECS 6699 Mathematics of Deep Learning - Final Report

May 2023

Tong Wu, tw2906

Yu Wu, yw3748

Abstract

In this report, we implemented ResNet-18 Neural Network using TensorFlow. We reproduced the result produced from paper, which compare the performance of several optimizers, containing Adam, Adam-W, Nadam, AMSGrad and SGD (Garg et al., n.d.). We reproduced a remarkably similar result as the original paper and analysis each optimizer's performance from mathematical aspect. In the next stage, we introduced Padam optimizer, which is proposed in 2021 and verified has higher robustness and accuracy than Adam and AMSGrad (Chen & Gu, 2022). We successfully implemented Padam optimizer into ResNet-18 Network and analysis its performance.

For the full source-code of our work, please refer to https://github.com/TongWu/EECS-E6699_Performance_of_ResNet_Architecture

1. Introduction

Convolutional Neural Networks (CNNs) have emerged as a crucial component in various applications because they can process and analyze visual and auditory data. Their unique prowess in handling images and signals has made them indispensable in many disciplines.

At the heart of CNNs lies their hierarchical structure, allowing them to automatically and adaptively discern patterns and relationships within the spatial arrangement of features in raw input data (Lecun et al., 1998). This innate ability to learn from data without explicit programming grants CNNs the power to extract relevant features accurately and efficiently from complex input representations.

By leveraging local connectivity and parameter sharing, CNNs can efficiently capture spatial information, making them particularly well-suited for tasks involving images or signals. The local connectivity in CNNs reduces the number of parameters, mitigating overfitting and enhancing generalization. Additionally, the parameter-sharing scheme enables the same convolutional filter to be applied across the entire input, facilitating the detection of the same feature irrespective of its position in the input data (Ankile et al., 2020).

Furthermore, the hierarchical structure of CNNs facilitates learning increasingly abstract and higher-level features as the network becomes deeper. This attribute allows CNNs to build robust and meaningful representations of the input data.

The Residual Network, ResNet, was designed to address the vanishing gradient problem for train the very-deep neural networks (He et al., 2015). The key improvement in ResNet is the use of residual connections (He et al., 2015). These connections allow the ResNet to learn residual functions, which are added to the input of the current neural network layer. Essentially, the network learns the difference between the input and the desired output (residual) instead of learning the output directly (He et al., 2015). This approach makes the ResNet architecture easier to learn identity mapping, which prevents the vanishing gradient problem.

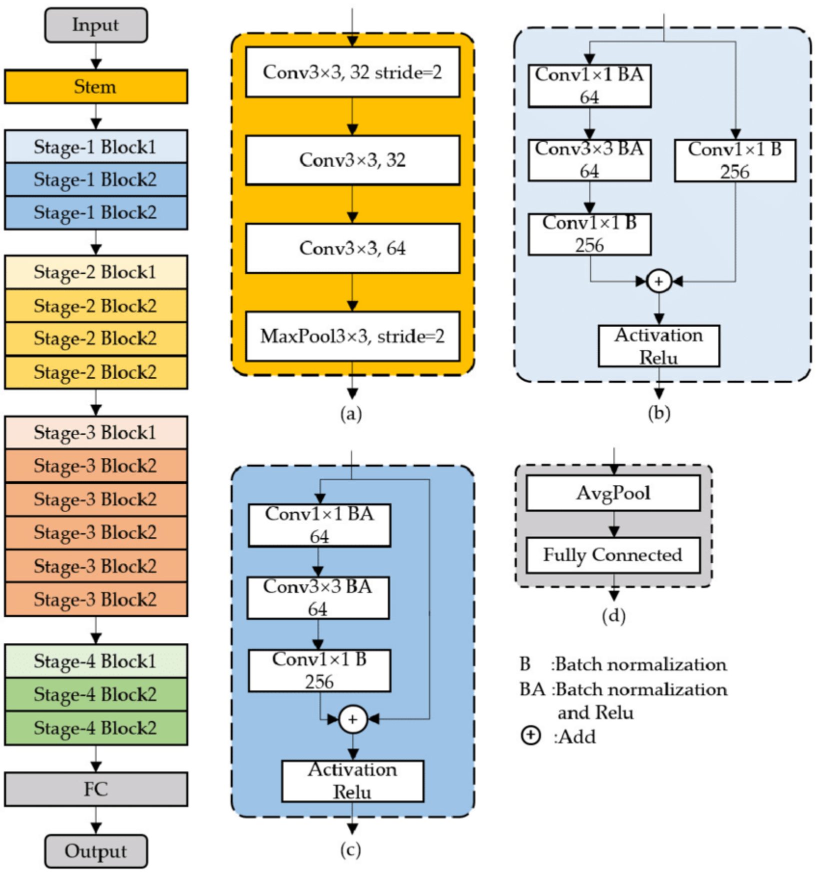


Figure 1 Architecture of ResNet-50

In a neural network model, the optimizer plays a crucial role in determining the model's accuracy, stability, and convergence rate. Its primary function is to modify the model's parameters to minimize the loss function's value. The optimizer employs the gradient descent method or its variations to update the model parameters, reducing the loss function's value and improving the model's fit on the training data.

In this project, we use stochastic gradient descent (SGD) optimizers. During each iteration of the training process, the SGD randomly selects a mini-batch of samples from the training set and calculates the gradients of the loss function concerning model parameters. The gradient the SGD optimizers calculated represents the direction in which the loss function increases fastest for the current parameter value. After using the computed gradient, the SGD optimizer updates the model's parameters. The update equation of the SGD optimizer is shown below:

$$\theta = \theta - \eta * \nabla L(\theta)$$

The learning rate is a hyperparameter that controls the parameter update step size. A more significant learning rate may lead to an unstable optimization process, while a smaller learning rate may lead to a slow convergence rate. SGD gradually optimizes the model parameters through several iterations. Each iteration uses a new mini-batch to compute the gradient and update the parameters. The iterative process continues until a predetermined number of iterations is reached, or other convergence conditions are met.

2. Problem Description

We implemented various leading optimizers to the ResNet neural network and compared their performance difference. In past papers and studies, we found that few papers analyze and compare multiple optimizers in one neural network architecture. It is essential to compare the performance between optimizers, and there following reasons:

1. Various optimizers may have different performances on different problems and model structures. Some optimizers may converge faster on some problems and perform poorly on others.
2. Different optimizers may have different convergence speeds. A fast converging optimizer can achieve similar performance in fewer training iterations, thus saving computational resources and time.
3. Different optimizers may have different sensitivities to hyperparameter selection, parameter initialization, and noisy data during training. Some optimizers are more robust in the face of these challenges.
4. By comparing the performance of different optimizers, we can gain more insight into the strengths and weaknesses of each optimizer and their applicability to specific problems.

3. Reproduction

In this project, we first need to replicate some experimental results. We chose a paper that explains in detail the performance of each optimizer in the ResNet-18 neural network model (Garg et al., n.d.). They trained and compared the performance of SGD, Adam, and AMSGrad using the CIFAR-10 dataset. In this chapter, we replicated that paper and introduced two different Adam variants, NAdam and Adam-W.

3.1 Experiment Setup

We use Google CoLab to train and test the model in the reproduction and the following new result part. All experiments use an Intel Xeon Silver 4214 Processor and one Nvidia A-100 GPU with 40GB of graphic memory.

3.2 Optimizers

SGD:

The SGD (Stochastic Gradient Descent) optimizer is a variant of gradient descent where the model parameters are updated using a random subset of training samples instead of the entire dataset. This stochastic nature of SGD can help in faster convergence and escape local minima and saddle points in non-convex optimization problems. SGD is particularly effective in training large-scale models on massive datasets, as it is computationally efficient compared to the standard gradient descent algorithm. The update rule for the model parameters can be written as follows:

$$g_t = (1/b) * \Sigma\,(gradient\ of\ loss\ function\ w.r.t.\theta\ for\ each\ sample\ in\ B)$$

$$\theta_t = \theta_{t-1} - \alpha * g_t$$

Adam:

The Adam (Adaptive Moment Estimation) optimizer is a popular and widely-used optimization algorithm. Adam is designed to adaptively update model parameters' learning rates based on their importance, which can result in faster convergence and better performance when training deep neural networks (Kingma & Ba, 2017).

Adam combines the advantages of two other optimization algorithms, AdaGrad and RMSProp (Kingma & Ba, 2017). It maintains separate adaptive learning rates for each model parameter using the first and second moments of the gradients. The update rule for the model parameters can be written as follows:

1. Update the biased first-moment estimate:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$

2. Update the biased second-moment estimate:

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$$

3. Correct bias in the first moment estimate:

$$\widehat{m}_t = m_t / (1 - \beta_1^t)$$

4. Correct bias in the second moment estimate:

$$\widehat{v_t} = v_t / (1 - \beta_2^t)$$

5. Update the model parameters:

$$\theta_t = \theta_{t-1} - \alpha * \widehat{m_t} / (\sqrt{\widehat{v_t}} + \varepsilon)$$

Nadam:

The Nadam optimizer (Nesterov-accelerated Adaptive Moment Estimation) combines the Adam optimizer and Nesterov Accelerated Gradient (NAG) methods. Nadam aims to improve the convergence rate and performance of the ADAM optimizer by incorporating the momentum of Nesterov's method (Dozat, 2016). Integrating Nesterov momentum helps the optimizer achieve faster convergence and more stable training. The main difference in the update rule between Nadam and Adam is that Nadam calculates the Nesterov moment to update the model parameters. The update rule for the model parameters can be written as follows:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$$
$$\widehat{m}_t = m_t / (1 - \beta_1^t)$$
$$m_{t\_nesterov} = \beta_1 * \widehat{m}_t + (1 - \beta_1) * g_t$$
$$\widehat{v_t} = v_t / (1 - \beta_2^t)$$
$$\theta_t = \theta_{t-1} - \alpha * m_{t\_nesterov} / (\sqrt{\widehat{v_t}} + \varepsilon)$$

Adam-W:

The Adam-W optimizer is an Adam optimizer variant directly incorporating weight decay into the optimization algorithm. Adam-W aims to address the issues with traditional weight decay regularization in Adam, mainly when learning rate schedules are used (Loshchilov & Hutter, 2019). Weight decay is a regularization technique used to prevent overfitting by adding a penalty term to the loss function based on the magnitude of the model parameters (Loshchilov & Hutter, 2019). The update rule for the model parameters can be written as follows:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$$
$$\widehat{m}_t = m_t / (1 - \beta_1^t)$$
$$\widehat{v_t} = v_t / (1 - \beta_2^t)$$
$$\theta_{t_{weightdecay}} = \theta_{t-1} - \lambda * \theta_{t-1}$$
$$\theta_t = \theta_{t_{weightdecay}} - \alpha * \frac{\widehat{m_t}}{\left(\sqrt{\widehat{v_t}} + \varepsilon\right)}$$

AMSGrad:

The critical difference between Adam and AMSGrad is that AMSGrad uses a maximum of all past squared gradients to update the second-moment estimate (Phuong & Phong, 2019).

This modification addresses the issues with the Adam optimizer's convergence properties, especially in the presence of sparse gradients and non-convex optimization problems. AMSGrad aims to improve the convergence rate and stability of the training process compared to the original Adam optimizer. The update rule for the model parameters can be written as follows:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$$
$$\widehat{m}_t = m_t / (1 - \beta_1^t)$$
$$v_{t\_max} = \max(v_{t\_max}(t-1), v_t)$$
$$\theta_t = \theta_{t-1} - \alpha * \widehat{m_t} / (\sqrt{v_{t\_max}} + \varepsilon)$$

3.3 Dataset

We use CIFAR-10 to train the model. CIFAR is the Canadian Institute For Advanced Research, where the dataset was developed. The CIFAR-10 dataset consists of 60,000 32x32 colour images, divided into 10 different classes, with 6,000 images per class (*CIFAR-10 and CIFAR-100 Datasets*, n.d.). The classes are mutually exclusive and represent everyday objects like animals and vehicles. The landscape of the CIFAR-10 dataset is shown below:

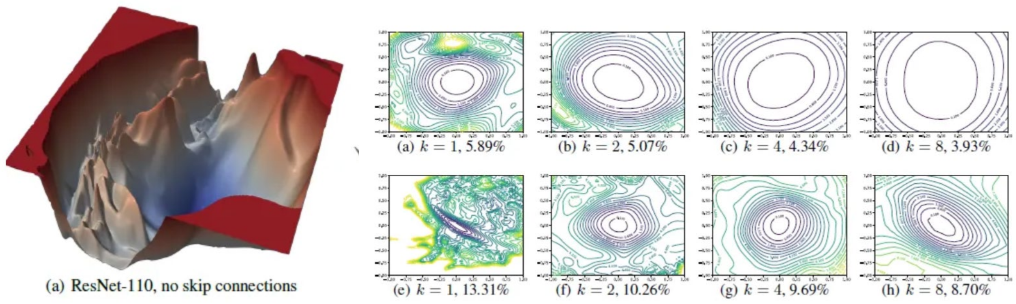
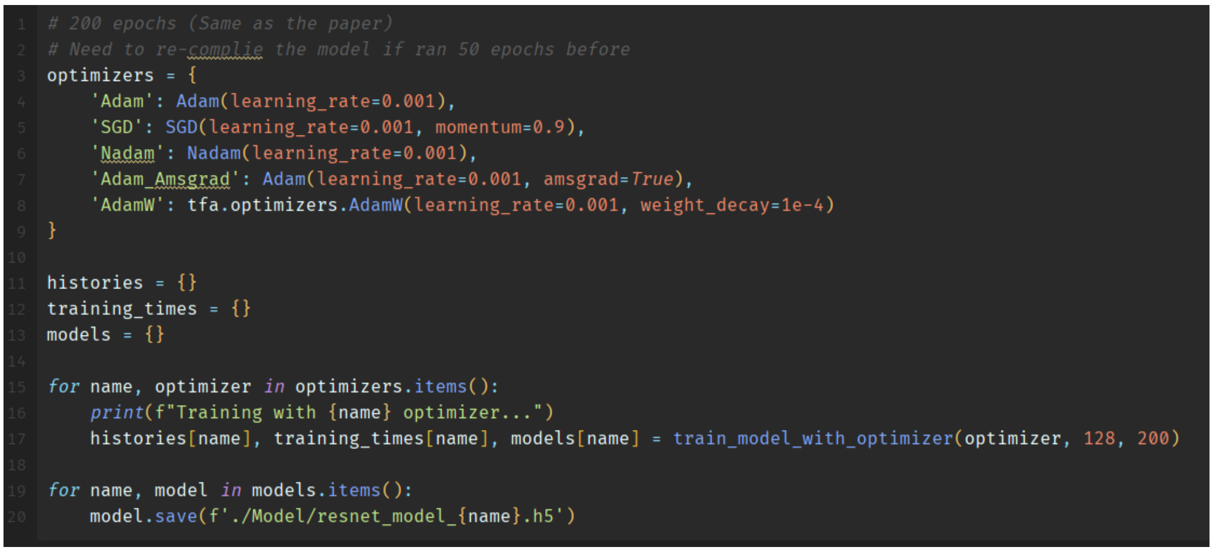


Figure 2 The loss surface and network width of ResNet-110 for CIFAR-10 (Tsang, 2023)

3.4 Method

In the reproduction process, we used ResNet-18 neural network, which contains 18 layers (excluding the activation function and pooling layers). The main structure of ResNet-18 consists of multiple convolutional layers, a ReLU activation function, residual blocks, a Batch Normalization layer, and a global average pooling layer. Finally, the different classes are a fully connected layer. In general, deeper layers mean the network has more hidden layers and can learn more complex and abstract feature representations. However, increasing the number of layers may lead to more severe gradient explosion and gradient disappearance problems, and the training time increased significantly.

The learning rate is a hyperparameter in the optimization algorithm that controls how quickly the model parameters are updated during the training process. The learning rate determines how much the model parameters should be changed after each iteration or each batch training to reduce the loss function. When setting up the optimizer, we uniformly setting the learning rate to 0.001. For the training parameters, we uniformly use 128 batches and have 200 epochs.



```python
# 200 epochs (Same as the paper)
# Need to re-complie the model if ran 50 epochs before
optimizers = {
    'Adam': Adam(learning_rate=0.001),
    'SGD': SGD(learning_rate=0.001, momentum=0.9),
    'Nadam': Nadam(learning_rate=0.001),
    'Adam_Amsgrad': Adam(learning_rate=0.001, amsgrad=True),
    'AdamW': tfa.optimizers.AdamW(learning_rate=0.001, weight_decay=1e-4)
}

histories = {}
training_times = {}
models = {}

for name, optimizer in optimizers.items():
    print(f"Training with {name} optimizer...")
    histories[name], training_times[name], models[name] = train_model_with_optimizer(optimizer, 128, 200)

for name, model in models.items():
    model.save(f'./Model/resnet_model_{name}.h5')
```

Figure 3 Implement Optimizers and Train the Model

3.5 Result and Analysis

After the training, we plot graphs containing all necessary performance indicators.

i. Training Loss

Training loss quantifies the divergence between the model's predictions and the true ground truth during the training stage. It is computed using a loss function, like mean squared error for regression tasks or cross-entropy loss for classification tasks.

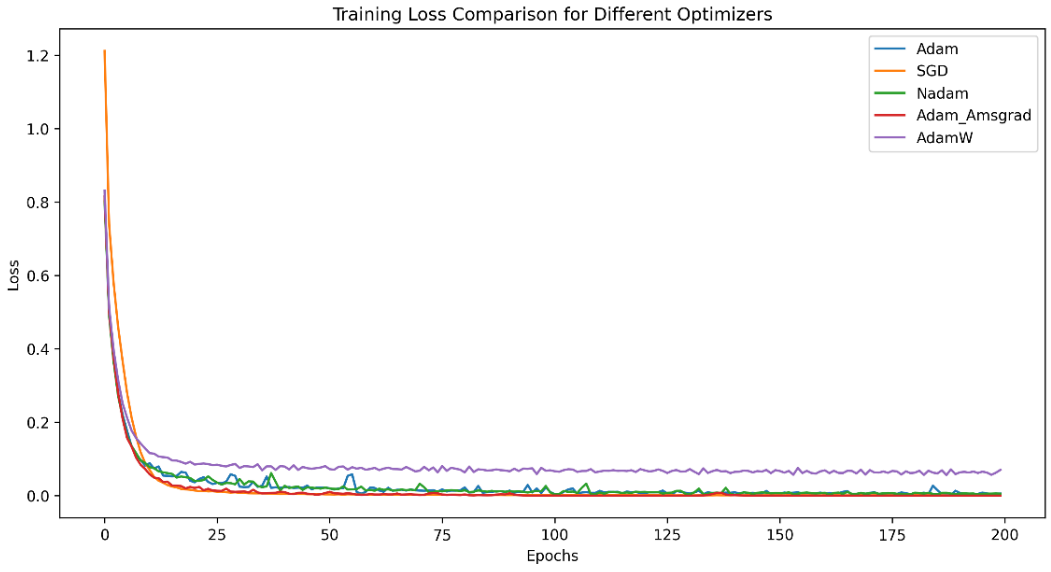


Figure 4 Training Loss of Optimizers

ii. Test Error

Test error measures the model's performance on a separate dataset not used during training, often called the test set. It is calculated similarly to the training loss, using a loss function or an error metric, such as mean squared error, cross-entropy loss, or classification error rate. The test error estimates how well the model generalized to unseen data. A lower test error indicates better generalization performance.

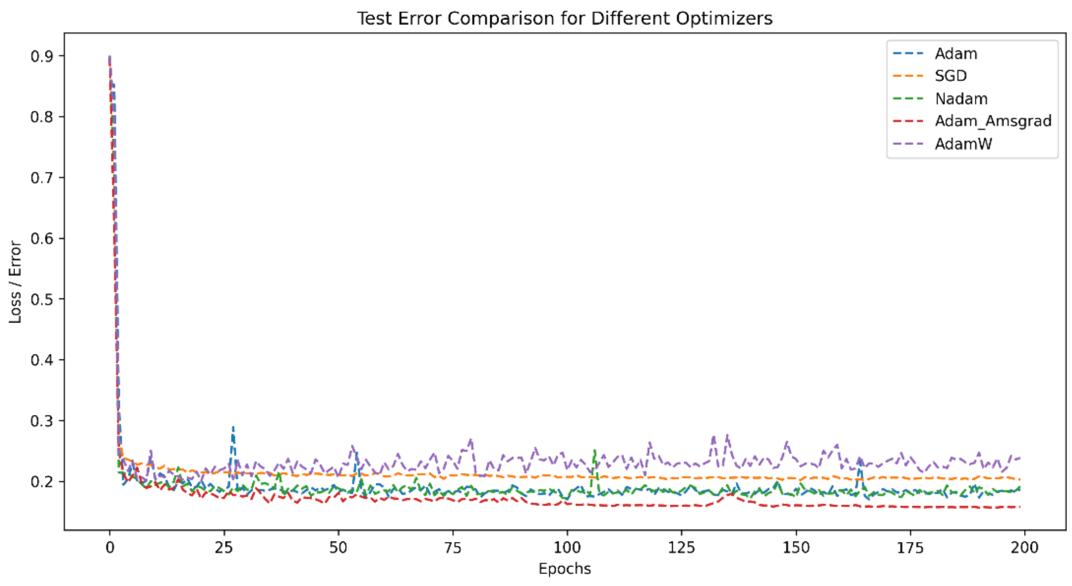


Figure 5 Test Error of Optimizers

iii. Validation Accuracy

Validation accuracy is a measurement utilized to evaluate a classification model's performance on an independent dataset not involved in the training process, known as the validation set. It represents the proportion of accurate predictions the model makes about the overall number of samples within the validation set. An increased validation accuracy signifies the enhanced performance of the model on the validation set.

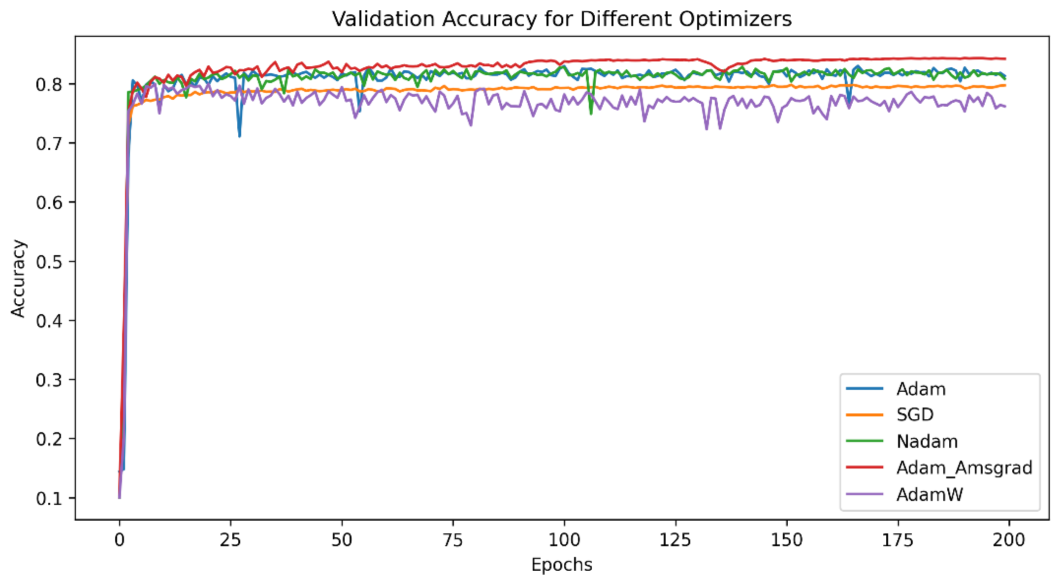


Figure 6 Validation Accuracy of Optimizers

iv. Training Time

Training time shows the time cost for each model's training.

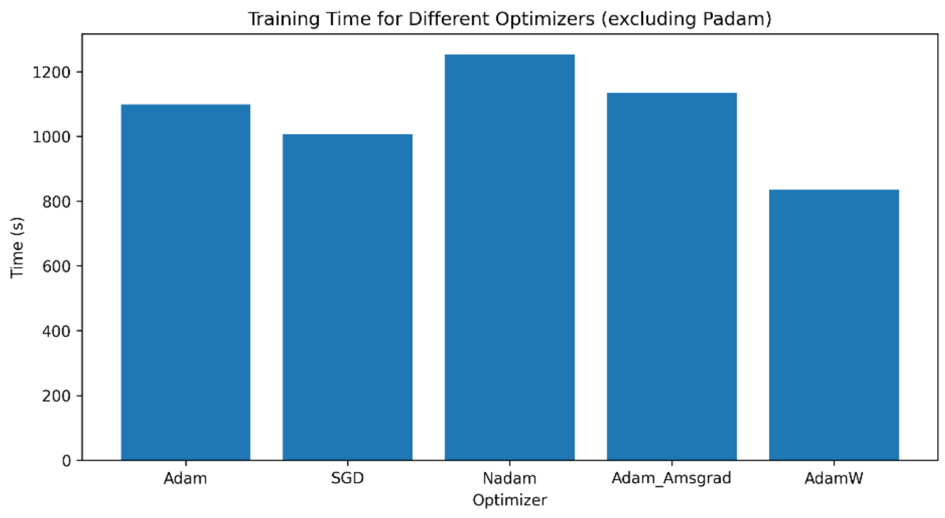


Figure 7 Training Time of Optimizers

v. Analysis

The graphs above show we reproduce the result. The blue line is Adam, and green line is Nadam, they perform oscillations in each graph which shows that they may not stable compared with the AMSGrad in red line which has the best performance overall. The graph shows that they all converge the training loss and test error into a meagre value lower than 20% and maintain accuracy around 75-80%. The SGD and Adam-W have the highest test error and lowest accuracy but the shortest training time.

Adam computes the adaptive learning rates for each parameter. Adam has high performance with low training loss and error. However, the learning rate of Adam may decrease too quickly when the second moment of gradients decreases in the later stages of training, which lets Adam to lose some robustness.

Nadam introduces an extra bias-correction step to reduce oscillations during training. In the graph, it leads to similar performance compared to Adam but with better convergence speed and robustness.

AMSGrad retains the maximum of all past-second moment estimates to prevent sudden drops in the learning rate, resulting in better optimization performance. Also, it uses a correction factor to ensure the gradient average is not underestimated during the iteration to get a better convergence performance. In general, AMSGrad has the best performance and robustness. However, it needs more time to compute.

As an Adam optimizer with improved weight decay processing, Adam-W usually performs better than the Adam optimizer. However, it may be less desirable in ResNet architectures. The improvement of Adam-W is mainly in the handling of weight fading. However, this treatment may not apply to all network architectures, such as ResNet, and the default hyperparameters may not maximize performance. But in general, Adam-W has the fastest training time.

SGD only compute the loss function according to the gradient to update the parameter, so the computing time of SGD is shorter than others. However, it doesn't have dynamic adjusting of the learning rate, so SGD has bad performance than other adaptive optimization algorithms.

4. New Results – Padam

Padam (Partial adaptive moments) is an optimization algorithm that combines the benefits of adaptive learning rate methods like Adam with the robustness of non-adaptive methods like SGD. The main idea behind Padam is to introduce a new hyperparameter, 'p', that controls the degree of adaptivity in the learning rate. The p-value allows for a smoother transition between adaptive and non-adaptive optimization methods, depending on the chosen value of 'p'.

4.1 Mathematical Expression

Padam extends the Adam optimizer by introducing the 'p' parameter. The update rule for Padam can be expressed as follows:

1. Calculate the gradient for the current iteration:

$$g_t = \nabla f(\theta_t)$$

2. Update biased first-moment estimate:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$

3. Update biased second raw moment estimate:

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$$

4. Compute bias-corrected first-moment estimate:

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

5. Compute bias-corrected second raw moment estimate:

$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

6. Update the parameters:

$$\theta_{t+1} = \theta_t - \alpha_t * \frac{\widehat{m_t}}{|\hat{v}_t|^p}, \text{where } \hat{v}_t = \max(\widehat{v_{t-1}}, v_t)$$

4.2 Advantage and Optimization

The 'p-value is a hyperparameter that determines the degree of adaptation of the learning rate for each parameter in the model. It ranges from 0 to 1, with 0 representing a non-adaptive update (i.e., using a fixed learning rate) and 1 corresponding to the original Adam algorithm (fully adaptive update) (Chen & Gu, 2022).

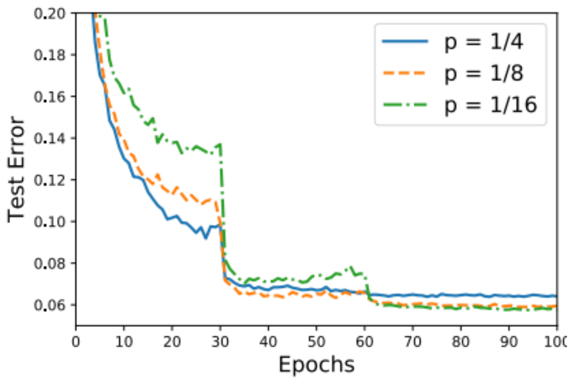
The performance of Padam depends on the choice of the p-value, which can affect the optimization process's convergence speed, robustness, and generalization ability. The optimal p-value depends on the problem and may vary for different tasks, models, and data sets. By introducing a 'p' hyperparameter, Padam can adapt the learning rate during training, which can help achieve faster convergence than Adam in some cases (Phuong & Phong, 2019). When 'p' is less than 1, the learning rate adaptation is less aggressive, allowing the optimizer to maintain a more significant learning rate during the initial training phase, thus achieving faster convergence. One of the limitations of Dam is that it can suffer from aggressive learning rate adaptation, which can lead to instability in training.

Padam addresses this issue by allowing the user to control the degree of adaptation using a p-value. A smaller p-value can lead to a more stable training process by reducing the aggressiveness of learning rate adaptation. The choice of the p-value can also affect the generalization ability of the trained model. The Padam optimizer with a suitable p-value can sometimes lead to better generalization performance than an Adam because it can balance adaptive and non-adaptive updates.

4.3 Dataset and Method

The experiment setup and used dataset are the same as the previous 'reproduction' part.

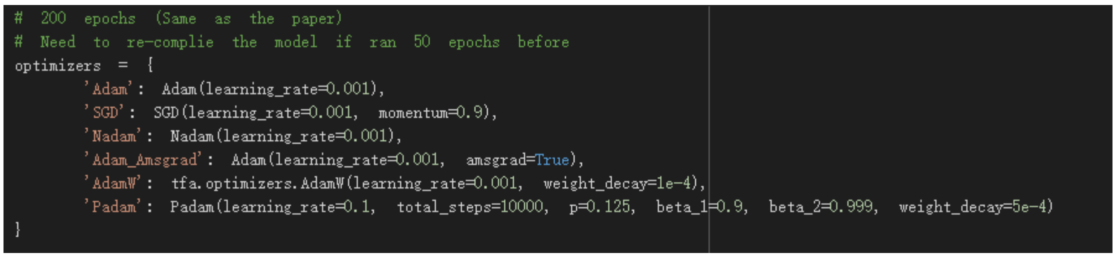
The original paper compares the performance of different 'p' values (Chen & Gu, 2022). From the paper, the test error decreased as the p-value increased. To show Padam's performance, we choose the intermediate value of 1/8.



(a) CIFAR-10

Figure 8 Test Error of Padam Various From 'p' Value

In this experiment, the goal is to compare the performance of the Padam and other SGD-based optimizers. The previous optimizers like Adam, Nadam, Adam-W, AMSGrad and SGD used the same hyperparameters as the previous reproduction part.



```
# 200 epochs (Same as the paper)
# Need to re-complie the model if ran 50 epochs before
optimizers = {
        'Adam': Adam(learning_rate=0.001),
        'SGD': SGD(learning_rate=0.001, momentum=0.9),
        'Nadam': Nadam(learning_rate=0.001),
        'Adam_Amsgrad': Adam(learning_rate=0.001, amsgrad=True),
        'AdamW': tfa.optimizers.AdamW(learning_rate=0.001, weight_decay=1e-4),
        'Padam': Padam(learning_rate=0.1, total_steps=10000, p=0.125, beta_1=0.9, beta_2=0.999, weight_decay=5e-4)
}
```

Figure 9 Code for Implementing Optimizers

4.4 Result and Analysis

After the training, we plot graphs containing all necessary performance indicators.

i. Training Loss

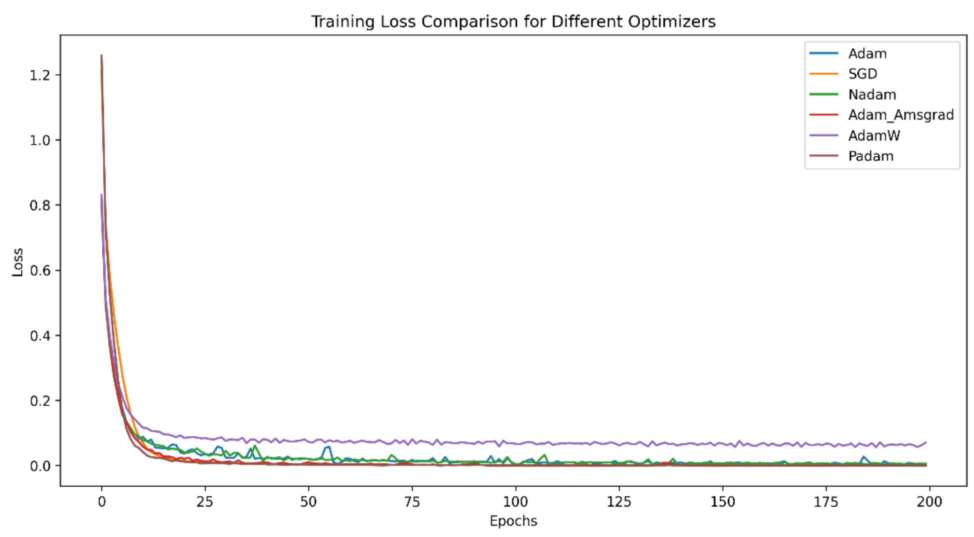


Figure 10 Train Loss of Optimizers

ii. Test Error

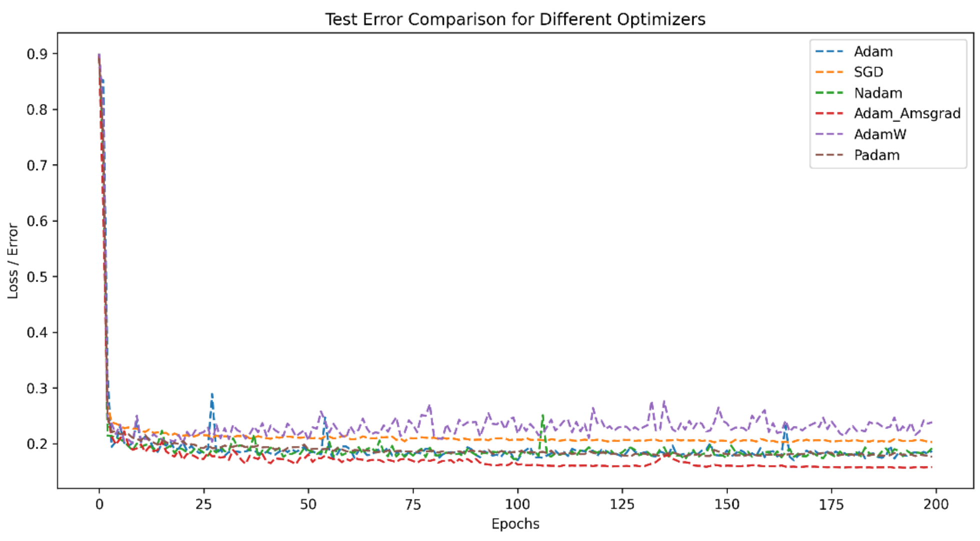


Figure 11 Test Error of Optimizers

iii. Validation Accuracy

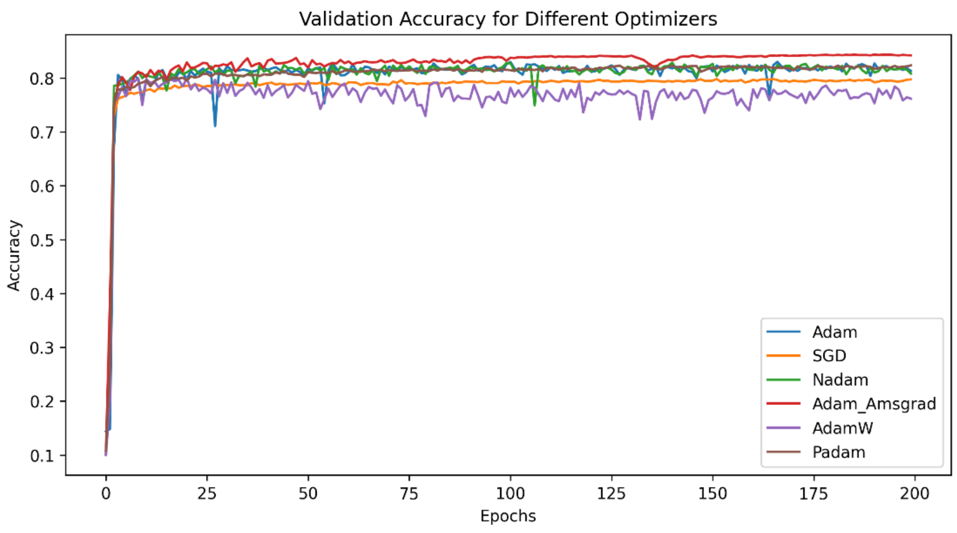


Figure 12 Validation Accuracy of Optimizers

iv. Training Time

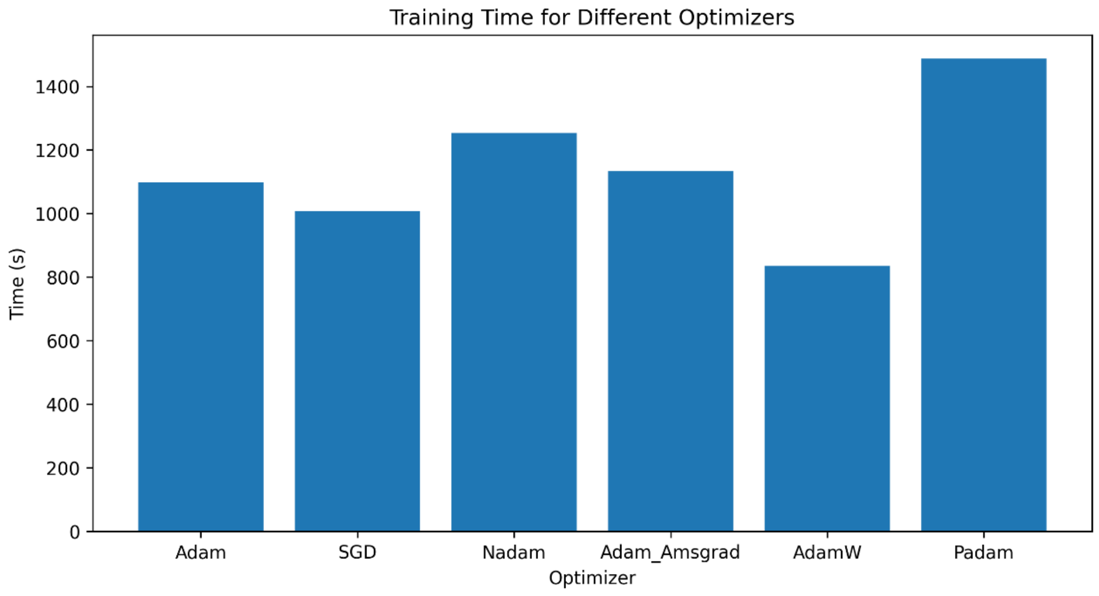


Figure 13 Training Time of Optimizers

v. Analysis

Padam introduces an additional hyperparameter, ' momentum reservoir', which controls the accumulation of past gradients in the optimizer, which can help the optimizer to converge faster to a better solution. However, setting this hyperparameter too high may lead to poor convergence, while setting it too low may result in slow convergence.

Due to its adaptive momentum reservoir mechanism, Padam may have a smoother test error data line compared to other optimizers, such as Adam and SGD. Padam updates the momentum reservoir based on the variance of the gradients, which can effectively smooth out the variations in the gradient updates and make the optimization process more stable.

However, due to the addition of the calculation and the fusion of the feature of AMSGrad and Adam, the training time of Padam is significantly higher than other optimizers.

5. Conclusion & Future Works

In this project and report, we implemented ResNet-18 Neural Network using TensorFlow. We reproduced the paper's result, which compares the performance of several optimizers containing Adam, Adam-W, Nadam, AMSGrad and SGD using the CIFAR-10 dataset (Garg et al., n.d.). We reproduced a remarkably similar result to the original paper and analyzed each optimizer's performance mathematically. In the next stage, we introduced Padam optimizer, which was proposed in 2021 and verified to have higher robustness and accuracy than Adam and AMSGrad (Chen & Gu, 2022). We successfully implemented the Padam optimizer into ResNet-18 Network and analyzed its performance. Although the model we implemented did not precisely match the original result from the original paper, it may be due to the architecture and hyperparameters tunning difference. However, Padam still brings an excellent performance compared to the others. In this project, we compared the performance of multiple optimizers in ResNet-18. However, a single neural network structure and a single dataset do not fully demonstrate the performance of the optimizers. Since each different optimizer has different performance in different architectures and datasets, it would be helpful to compare performance in different architectures (e.g., VGGNet, DenseNet, GAN) and compare performance in different datasets (e.g. SVHN, ImageNet, STL10). Then it will provide more exciting and comprehensive results. We have written source code to compare performance in VGGNet-16 and DenseNet-121 architectures. However, due to the enormous computational resources and training time required for these two architectures, we had to suspend the training and only provide the source code, leaving it as future work.

6. References

Ankile, L. L., Heggland, M. F., & Krange, K. (2020). *Deep Convolutional Neural Networks: A survey of the foundations, selected improvements, and some current applications* (arXiv:2011.12960). arXiv. https://doi.org/10.48550/arXiv.2011.12960

Chen, J., & Gu, Q. (2022). *Padam: Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks*. https://openreview.net/forum?id=BJll6o09tm

CIFAR-10 and CIFAR-100 datasets. (n.d.). Retrieved May 8, 2023, from https://www.cs.toronto.edu/~kriz/cifar.html

Dozat, T. (2016). *INCORPORATING NESTEROV MOMENTUM INTO ADAM*.

Garg, C., Garg, A., & Raina, A. (n.d.). *Analysis of Gradient Descent Optimization Algorithms on ResNet*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition* (arXiv:1512.03385). arXiv. https://doi.org/10.48550/arXiv.1512.03385

Kingma, D. P., & Ba, J. (2017). *Adam: A Method for Stochastic Optimization* (arXiv:1412.6980). arXiv. https://doi.org/10.48550/arXiv.1412.6980

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Loshchilov, I., & Hutter, F. (2019). *Decoupled Weight Decay Regularization* (arXiv:1711.05101). arXiv. https://doi.org/10.48550/arXiv.1711.05101

Phuong, T. T., & Phong, L. T. (2019). On the Convergence Proof of AMSGrad and a New Version. *IEEE Access*, *7*, 61706–61716. https://doi.org/10.1109/ACCESS.2019.2916341

Tsang, S.-H. (2023, February 1). Brief Review—Visualizing the Loss Landscape of Neural Nets. *Medium*. https://sh-tsang.medium.com/brief-review-visualizing-the-loss-landscape-of-neural-nets-dd93cb261afc