Algerian Democratic and Popular Republic
Ministry of Higher Education and Scientific Research

KASDI MERBAH UNIVERSITY OUARGLA
Faculty of New Information and Communication Technologies
Department of Computer Science and Information Technology

Thesis Submitted in Candidacy for a Master Degree in computer science, option

Artificial Intelligence and Data Science

by Khaoula Dahimi

Theme

Adam Variants Optimizers: A Comparative Study

Evaluation date: 22/06/2024

Jury Members:

Dr.	Kherfi Mohammed Lamine	Jury Chair	UKM Ouargla
Dr.	Khadra BOUANANE	Supervisor	UKM Ouargla
Dr.	Aiadi Oussama	Examiner	UKM Ouargla

Academic year: 2023/2024

Acknowledgments

Firstly, I am deeply indebted to Allah Almighty for His blessings and grace, without which this work would not have been possible, and my efforts would not have succeeded.

I extend my heartfelt appreciation to Dr. Khadra Bouanane, my supervisor, for her invaluable guidance and unwavering support throughout the completion of our thesis.

I would like to express my sincere gratitude to my partner, Basma Dokkar, for their invaluable assistance in this work.

Lastly, my deepest gratitude goes to my parents, siblings, and friends for their unwavering support and encouragement throughout this journey. Their understanding and patience during the long hours of work were instrumental in my achievements

Dedication

This thesis is dedicated to my family members, whose unwavering love and support have been the driving force behind my academic pursuits. Their encouragement and belief in me have been a constant source of inspiration.

I also dedicate this work to my teachers and my supervisor, Dr. Bouanane, whose guidance made the completion of this thesis possible.

ABSTRACT

The Adam optimizer is widely regarded as a highly effective algorithm for training deep learning models. However, there are instances where Adam may not perform optimally and could lead to poor generalization. To address these limitations, several variants of Adam have been introduced to mitigate its drawbacks. For example, AdaMod, AdaBound, R-Adan and N-Adam. this thesis aims to introducing the best optimizer and which Adam limitations addressed that has impact to its performance by providing comparative study of these variants include AdaMod, Adabound, R-Adam, AdaMax, AMSGrad, AdaBeleif, E-Adam, YOGI, AdamW, N-Adam, ND-Adam, MSVAG, T-Adam, and Ro-Adam. The results shows that when we employ a basic CNN and ResNet34 trained on the MNIST and CIFAR10 datasets respectively, AdaMod and AMSGRAD achieves the highest performance with accuracy scores $98.8\%, 73.4\%$ respectively. These optimizers behave similarly and achieve nearly the same results. To better understand and ensure the selection of the best optimizer, we utilized an LSTM architecture to train a sentence completion model on the Penn Treebank dataset and a time series forecasting model using the Amazon stock dataset. We found that AdamW and T-Adam outperformed the other optimizers.

Key words: Adam, Adam variants, optimizers, comparative study.

ملخص

تعتبر Adam من أفضل الخوارزميات استعمالا في التعلم العميق بسبب سرعة ادائها. إلا انها احيانا تقدم اداء ضعيفا بسبب عدة مشاكل تواجهها أثناء التدريب. و لمعالجة هذة المشاكل تم طرح عدة تعديلات بخصوص Adam و لتحسين ادائها و سرعتها مثل AdaMod, AdaBound, R-Adam . تهدف هذه الاطروحة الى ايجاد افضل خوارزمية و تحديد مشاكل Adam التي لها تأثير كبير على ادائها من خلال دراسة مقارنة لهذه الخوارزميات AdaMod, AdaBound, R-Adam, AdaMax, AMSGrrad, AdaBelief, E-Adam, YOGI, AdamW, N-Adam, ND-Adam, MSVAG, T-Adam, Ro-Adam . بينت النتائج أنه عند تدريب CNN و ResNet34 باستعمال البيانات MNIST , CIFAR10 أن AdaMod و AMSGrad حققا أفضل أداء بدقة بلغت 98.8% و 73.4% . لوحظ ايضا ان النتائج كانت متقاربة جدا. لذا لفهم أعمق حول أداء هذه الخوارزميات استعملنا LSTM مدرب على نوعين من البيانات Penn Treebank و Amazon stock . ولقد وجدنا أن AdamW و T-Adam قدما أداء عالي و تفوقا على الأخرين .

الكلمات المفتاحية: التحسين من الدرجة الاولى، Adam,

Contents

List of Figures

List of Tables

General Introduction

1 Introduction

Optimization is the backbone of deep learning, ensuring that neural networks learn efficiently and effectively from data. Fundamentally, it's the process of fine-tuning the parameters of a neural network to minimize the difference between the predicted output and the actual output[19].

Optimization methods are crucial due to the widespread success of deep learning models across various domains. However, the complexity of these models and the vast amounts of data have made the optimization process challenging. This complexity necessitates advanced optimization algorithms that can effectively handle large datasets and intricate model architectures[36].

The first-order optimization methods have become the most commonly used algorithms for training deep learning models due to their high performance. These methods can be categorized into two classes: the first class includes methods with a fixed step size towards the optimum, such as SGD[31] and NAG[27], while the second class includes methods that update the step size for each parameter, like AdaGrad[10], RMSProp[39], and Adam[6].

Adam is the default optimizer for many deep learning applications due to its stability and fast convergence. It combines the benefits of AdaGrad and RMSProp, maintaining their advantages. However, many studies have shown that Adam can exhibit poor generalization in certain scenarios, because of the extreme learning rates and sensitivity to noise, which can reduce its performance.

Several variants of Adam have been proposed to overcome its limitations in different ways, including AdaBound[24], T-Adam[17], N-Adam[9], and R-Adam[20]. These modified versions have demonstrated their effectiveness in enhancing performance. While each variant addresses specific issues, the optimal choice of optimizer often depends on the particular use case, dataset, and model architecture. Therefore, empirical comparisons are crucial to identify the most suitable optimizer for a given scenario [4].

While many comparative studies focus on similar objectives, they typically include only one or a few variants of the Adam algorithm. For instance, the authors in [34, 37] included the five most popular algorithms—SGD, Adagrad, AdaDelta, RMSprop, and Adam—but did not use any Adam variants. In contrast, Dogo et al. [8] included two Adam variants, AdaMax and N-Adam, while Derya [35] included three Adam variants: AdaMax, N-Adam, and AMSGrad.

Our study aims to formally assess the selection of the optimal optimizer among 14 variants of Adam across three distinct tasks: image classification, language modeling, and time series forecasting with various data structures. The objective is to determine which approach significantly enhances performance in each task.

2 THESIS STRUCTURE

The format of our thesis is as follows:

The chapter 1 begins with an introduction to the models employed in our study. We delve into their descriptions and highlight their relevance to our research objectives. Following this, we emphasize the significance of optimization methods in the training process, discussing several widely adopted techniques. Lastly, we address the challenges encountered during model training, providing insights into the complexities and issues that arise.

In the chapter 2, we delve deeply into the Adam optimizer, discussing its intricacies and highlighting several of its limitations. Additionally, we present various modified versions of Adam, categorized based on their specific adjustments aimed at addressing different Adam challenges.

The chapter 3 details the essential experimental setup requirements for a comparative study across three tasks, encompassing data selection and preprocessing, model architecture, and evaluation metrics.

The outcomes of our experimental work are detailed in Chapter 4 of the study, where we provide a comprehensive presentation of the results. This chapter also includes an extensive discussion to analyze and interpret the findings in context.

CHAPTER 1

EXPLORING MODEL ARCHITECTURES AND LEARNING PROCESSES IN DEEP LEARNING

1 INTRODUCTION

The effectiveness of deep learning models relies primarily on three factors: data, model architecture, and the learning process. Firstly, the quality and quantity of data influence the model's ability to generalize and adapt to different tasks. Secondly, the design of the model's architecture determines its capacity to capture and extract meaningful features from the input data. Furthermore, the learning process, including optimization algorithms and training strategies, plays a crucial role in refining the model's parameters and improving its performance.

In this chapter, we focus on two key factors: model architectures and the learning process. First, we provide a brief overview of several architectures utilized across various tasks relevant to this work. Next, we shift our attention to the learning process, with particular emphasis on the optimization methods used to train deep learning models. Finally, we discuss potential challenges encountered during the training process and strategies to address them.

The figure 1.1 illustrates the three factors that influence the performance of a deep learning model.

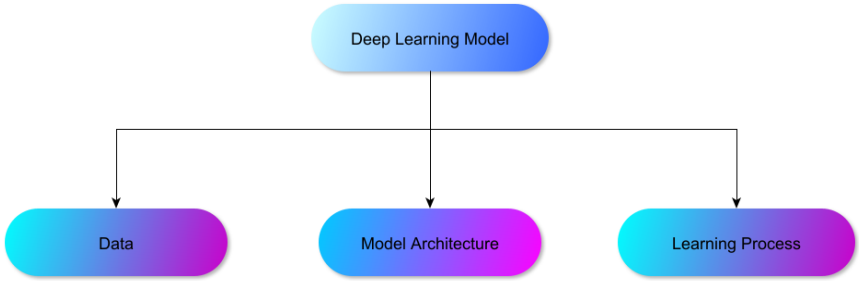

Figure 1.1: Model training factors

2 MODEL ARCHITECTURE

Within the domain of deep learning, a model denotes an architectural or framework designed to process and learn from data. It consists of layers of interconnected nodes, structured in a specific configuration to perform various tasks[2].

The following models are the primary focus of this study and will be briefly explained in the next subsection.

2.1 CONVOLUTIONAL NEURAL NETWORK

A Convolutional Neural Network (CNN)[41, 42] is a type of deep neural network commonly used in computer vision tasks such as image classification, object detection, and image segmentation. CNNs are designed, as depicted in the figure 1.2, to automatically and adaptively learn spatial hierarchies of features from input images through the use of convolutional layers. The key components and concepts of CNNs:

Convolutional Layers: are the core building blocks of CNNs. They consist of learnable filters (also called kernels or weights) that are convolved with the input image to produce feature maps. Each filter detects specific patterns or features, such as edges, textures, or object parts.

Pooling Layers: are often used after convolutional layers to reduce the spatial dimensions of feature maps while retaining important information.

Fully Connected Layers: also known as dense layers, are typically placed at the end of the CNN architecture. These layers connect every neuron in one layer to every neuron in the next layer, enabling high-level feature representation and classification.

Activation Functions: introduce non-linearities into the network, allowing it to learn complex mappings between inputs and outputs. For example RELU, Sigmoid and tanh.

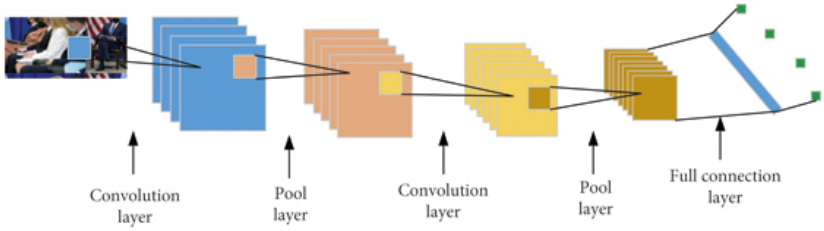

Figure 1.2: CNN Architecture

2.2 RESNET

ResNet[15], short for Residual Network, is a deep neural network architecture shown in the figure 1.3, that was designed to address the problem of vanishing gradients in very deep neural networks, which can hinder training and limit the network's ability to learn complex patterns. ResNet model consists of several residual blocks, each containing multiple convolutional layers, batch normalization, ReLU activation functions and shortcut connections. ResNet variants like ResNet50, ResNet34 have been widely used in various computer vision tasks, such as image classification, object detection.

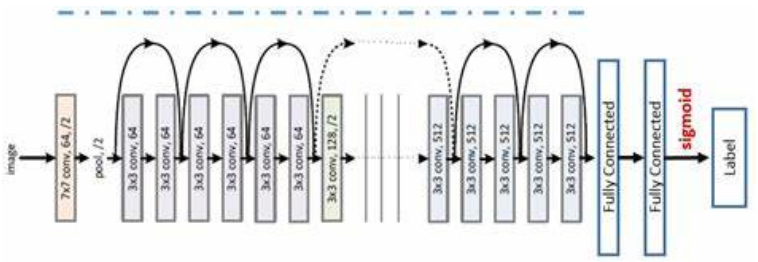

Figure 1.3: ResNet Architecture

2.3 LONG SHORT-TERM MEMORY

LSTM [16, 48], a form of recurrent neural network (RNN) architecture, addresses the vanishing gradient problem and captures long-term dependencies within sequential data.

As a reminder, RNNs [42, 48] are designed for processing sequential data, functioning as connectionist models that capture sequence dynamics via cyclic patterns within their node network. They maintain a state capable of encoding information from a context window of any length, enabling them to theoretically map from the entire history of previous inputs to each output.

Each RNN comprises multiple instances (across time) of the same network, with each instance conveying information to the subsequent one. The trainable parameters of an RNN, including weights and biases, are shared across all time-steps.

Within the LSTM architecture, memory cells like in the figure 1.4, are accompanied by three essential gates: the input gate, the forget gate, and the output gate. These gates regulate the flow of information into and out of the memory cells, enabling the network to remember or forget information selectively.

Input Gate : is responsible for deciding which information from the input should be stored within the memory cell.

Forget Gate: determines which information to discard from the cell state

Output Gate: controls what information should be output from the memory cell to the rest of the network

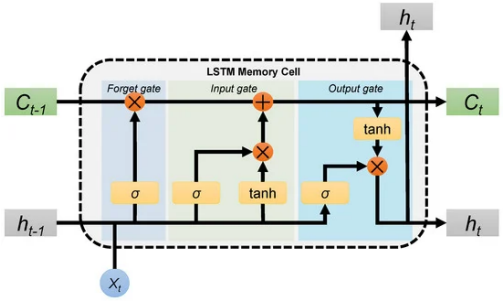

Figure 1.4: LSTM Unit Cell Architecture

3 LEARNING PROCESS

Learning process refer to the process of training a deep learning model for a specific task. It involves a series of organized steps such as selecting a loss function, choosing an optimization algorithm for parameter adjustment, and evaluating the model. For the purpose of this work, we specifically emphasize the optimization algorithms in deep learning.

Optimization methods are crucial in training deep learning models. Their importance can be presented by:

Improved Performance: Optimization algorithms play a pivotal role in enhancing the performance of deep learning models by diminishing errors and increasing accuracy. Through the identification of the optimal parameter set, the model becomes more adept at capturing intricate patterns within the data [12].

Faster Convergence Efficient optimization techniques speed up the learning process by reaching an optimal solution in fewer iterations. This is important for training deep learning models with extensive datasets, saving both time and computational resources[11].

Robustness: Well-optimized models are more robust to variations and noise in the data [3].

DL models often have millions of parameters, making the optimization problem highly complex. Hence, iterative algorithms are used to dissect this complexity, breaking down the optimization task into manageable steps[40]. These algorithms iteratively adjust the model's parameters to minimize the loss function, using gradients that signify the steepest descent direction in the loss landscape. Consequently, they guide the optimization process toward attaining the optimal solution.

4 GRADIENT DESCENT OPTIMIZATION ALGORITHMS

Gradient descent[32] is one of the most popular algorithms to perform optimization in deep learning. It's an algorithm that minimize an objective function $f(\theta)$, where θ repre-

sents the parameters of the model in D-dimensional space, by iteratively updating these parameters in the opposite direction of the gradient of the objective function $\nabla_\theta f(\theta)$ with respect to θ.

It uses the following formula (known as the updating rule) to adjust the parameters θ where The learning rate, denoted by α, determines the magnitude of the steps we take to reach a (local) minimum.

$$\theta_t = \theta_{t-1} - \alpha \cdot \nabla_\theta f(\theta_t) \tag{1.1}$$

Gradient descent methods encompass two complementary categories essential for optimizing deep learning models:

Data-aware Gradient Calculation: These methods adapt gradient computations based on the volume and characteristics of data employed. The trade-off between them is the accuracy of the gradient versus the time complexity to perform each parameter's update.

Parameter Update Strategies: These methods leverage gradients to iteratively adjust model parameters during training, aiming to seep up the convergence and improve the model performance.

These categories collaborate during training to ensure efficient gradient computation and optimal parameter adjustments, leading to enhanced model optimization. For further details, refer to the figure 1.5.

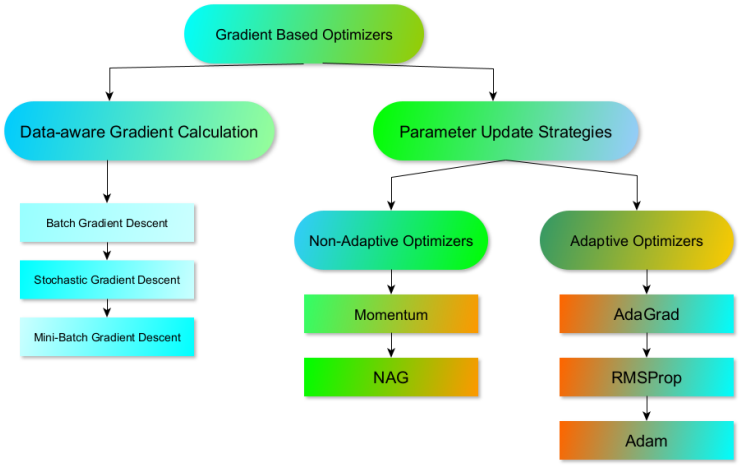

Figure 1.5: Gradient Descent Optimization Algorithms

4.1 DATA-AWARE GRADIENT CALCULATION

Gradient descent has three types based on the amount of data used to compute the gradient of the objective function. When dealing with large amounts of data, we balance the accuracy of parameter updates against the time required to complete them.

4.1.1 BATCH GRADIENT DESCENT

Batch gradient descent[32] uses the whole dataset to compute the gradient of the objective function at each iteration.

$$\theta_t = \theta_{t-1} - \alpha \cdot \nabla_\theta f(\theta) \tag{1.2}$$

This method provides correct parameter changes, however, it can be sluggish and computationally expensive, particularly for big datasets.

4.1.2 STOCHASTIC GRADIENT DESCENT

Instead of using the entire dataset to compute the gradient of the objective function $f(\theta)$, SGD uses only a single random example at each iteration[31].

$$\theta_t = \theta_{t-1} - \alpha \cdot \nabla_\theta f(\theta, x, y) \tag{1.3}$$

Where x is a sample from the dataset and y is its label.

This method introduces randomness into the parameter updates, which can help escape local minima and speed up convergence, especially in large datasets. However, it may result in more noisy updates (fluctuation) compared to batch gradient descent.

4.1.3 MINI-BATCH GRADIENT DESCENT

mini-batch gradient descent[32] computes the gradient of the objective function $f(\theta)$ using a mini-batch of data samples at each iteration.

$$\theta_t = \theta_t - \alpha \cdot \nabla_\theta f(\theta, X_i, Y_i) \tag{1.4}$$

Where X_i are data samples of ith batch and Y_i are its labels.

This method reduces the noisy updates, which can lead to more stable convergence and it strikes a balance between the accuracy of parameter updates and computational efficiency, making it widely used in practice for training neural networks.

4.2 PARAMETER UPDATE STRATEGIES

Also known as gradient descent optimizers, are adaptations of the gradient descent algorithm designed to mitigate its shortcomings. These methods tackle challenges such as selecting an appropriate learning rate or mini-batch size, both of which significantly impact the training process and the model's performance.

In the following, we will cover some of the commonly used optimizers in Deep Learning, categorized into two groups.

4.2.1 NON-ADAPTIVE METHODS

Non-adaptive methods in optimization refer to techniques that do not adjust the learning rate during training based on the observed behavior of the optimization process.

Momentum Is a technique for accelerating gradient descent by adding the momentum term which helps in smoothing out the noisy gradients[29].

$$v_t = \beta v_{t-1} + (1 - \beta)\nabla_\theta f(\theta_t) \tag{1.5}$$

$$\theta_t = \theta_{t-1} - \alpha v_t \tag{1.6}$$

Where v_t is the velocity,$\nabla_\theta f(\theta)$ is the gradient of the objective function at θ_t, α is the learning rate, β is the momentum hyper-parameter that controls the influence of past gradients on the current update.

By doing so, The momentum term increases for dimensions with consistent gradient directions and reduces updates for dimensions where gradients change direction. Consequently, this leads to fast convergence and diminished oscillation.

Nesterov Accelerated Gradient (NAG) In order to enhancing the effectiveness of Momentum, NAG[27] computes the gradient not at the current position θ_t but at an estimated

of the next position $\theta_t - \beta v_{t-1}$

$$v_t = \beta v_{t-1} + \alpha \nabla_\theta f(\theta_t - \beta v_{t-1}) \tag{1.7}$$

$$\theta_t = \theta_{t-1} - v_t \tag{1.8}$$

Such as v_t is the velocity,$\nabla_\theta f(\theta)$ is the gradients at θ_t, α the learning rate and β is the momentum term.

This approach often results in faster convergence and improved stability compared to stochastic gradient descent and Momentum.

Limitations Non-adaptive methods, while simpler and often easier to implement, present several limitations that require attention.

Selecting an appropriate learning rate poses a challenge. A learning rate that is overly small results in slow convergence, whereas one that is excessively large can hinde convergence and induce fluctuations or divergence in the loss function around the minimum[32]. The figure1.6 depicts the effect of the learning rate on the optimizer's behavior.

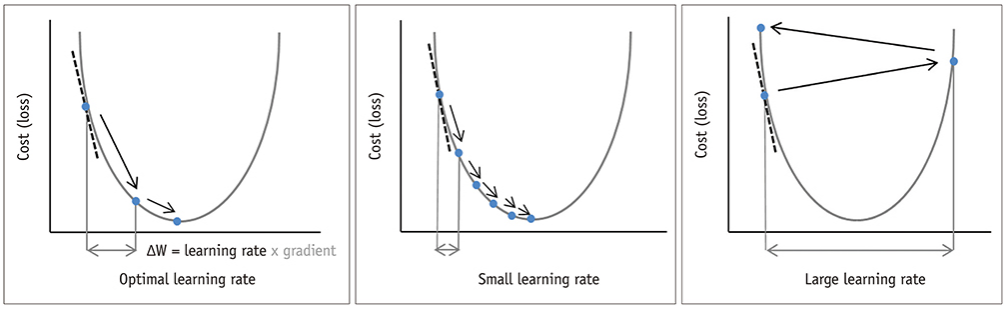

Figure 1.6: The impact of the Learning Rate

Learning rate schedules modify the training rates using techniques like annealing or a predefined strategy, reducing the rate when the objective function drops below a certain threshold. However, these schedules and thresholds are predetermined, which means they cannot dynamically adapt to the unique characteristics of the dataset being used. This limitation can reduce the effectiveness of the training process[5].

Furthermore, when employing a uniform learning rate, all parameter updates are subjected to the same rate. In scenarios where data is sparse and features exhibit varying frequencies, it may be preferable to adjust parameters differentially[32].

Addressing these limitations often involves moving towards adaptive methods that can adjust to varying conditions and provide more robust performance across different scenarios.

4.2.2 ADAPTIVE METHODS

Adaptive methods dynamically modify the learning rate throughout training, enabling them to adapt to the specific characteristics of both the optimization problem and the dataset.

Adaptive Gradient (AdaGrad) AdaGrad[10] is an algorithm for gradient-based optimization that customizes the learning rate based on parameter frequencies, executing larger updates for infrequent parameters and smaller ones for frequent ones. Consequently, it proves particularly effective in handling sparse data.

$$v_t = \sum_{i=0}^{t} \nabla_{\theta_t} f(\theta_{t,i})^2 \tag{1.9}$$

$$\alpha_t = \frac{\alpha_{t-1}}{\sqrt{v_t + \epsilon}} \tag{1.10}$$

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_t} f(\theta_{t,i}) \tag{1.11}$$

Such as v_t is the accumulated sum of squared gradients, α_t is the adaptive learning rate at t and ϵ is a small constant added to avoid division by zero.

AdaGrad may suffer from a diminishing learning rate problem. During training neural networks. The learning rate gets scaled down so much that the algorithm ends up stopping entirely before reaching the global optimum.

Root Mean Square Propagation (RMSProp) The RMSProp[39] optimizer shares similarities with AdaGrad but offers improvements, particularly in non convex scenarios. It achieves this by replacing gradient accumulation with an exponentially weighted moving average, leading to enhanced performance.

$$v_t = \beta v_{t-1} + (1 - \beta) \nabla_{\theta_t} f(\theta_{t,i})^2 \tag{1.12}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{v_t} + \epsilon} \nabla_{\theta_t} f(\theta_{t,i}) \tag{1.13}$$

where v_t is the moving average of the squared gradients, β is decay rate, α the learning rate and ϵ is a small constant.

Adaptive Moment Estimation (ADAM) Adam[6] combines the strengths of the AdaGrad and RMSProp optimizers to calculate adaptive learning rates for each parameter. This approach helps accelerate convergence and improve performance.

$$g_t = \nabla_\theta f(\theta_{t-1}) \tag{1.14}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1.15}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{1.16}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{1.17}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{1.18}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{1.19}$$

Where m_t and v_t are the first and the second moments, $\beta_1, \beta_2 \in [0, 1]$ are smoothing parameters that control m_t and v_t respectively. $\hat{m}_t$ and $\hat{v}_t$ are the bias correction of m_t and v_t respectively.

We provide a detailed explanation of Adam in the next chapter.

Gradient descent and its variants are widely adopted for improving model performance. However, during the training process, these methods encounter various challenges.

5 CHALLENGES DURING TRAINING PROCESS

Throughout the training of deep learning models, numerous challenges and issues may emerge, impeding the optimization process and influencing the model's efficacy. Some common problems include:

5.1 VANISHING GRADIENT

The issue of unstable gradients, commonly known as the **vanishing gradient** [28], poses a significant challenge in training neural networks. Specifically, it affects the weights in earlier layers of the network. During training, optimizer like Stochastic Gradient Descent (SGD) compute gradients of the loss function with respect to the model parameters. However, sometimes these gradients become exceedingly small, causing the optimizer to update the weights proportionally to the gradient's magnitude. If the gradient diminishes to a very small value, the weight update becomes negligible. Consequently, the updated weight barely deviates from its original value, offering minimal contribution to reducing the loss. This stunted progress results in the weight remaining stuck, unable to approach its optimal value. This stagnation affects subsequent layers of the network, hindering the overall learning process.

5.2 EXPLODING GRADIENT

Training very deep neural networks can encounter a challenge known as **exploding gradients** [28], wherein the derivatives or slopes become excessively large, complicating the training process. When gradients become extremely large, the optimizer's update step also becomes magnified, potentially resulting in detrimental parameter updates where values oscillate wildly or even become infinite (inf) or undefined (NaN). In such cases, training may necessitate restarting from a prior checkpoint.

A straightforward solution to mitigate exploding gradients is a technique known as **gradient clipping**. This approach involves setting a predefined threshold, a hyperparameter chosen by the user. If the norm of the gradient exceeds this threshold, it is scaled down before being applied in the optimizer update step. By clipping gradients to a manageable range, gradient clipping helps stabilize the training process and prevents excessively large updates that can lead to instability or convergence issues.

Algorithm 1 gradient clipping

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
if $\|g_t\| > threshold$ **then**
  $g_t \leftarrow \frac{threshold}{\|g_t\|} g_t$
end if

5.3 UNDERFITTING

Underfitting[26], a prevalent issue in machine learning, typically arises when the model hasn't been sufficiently trained with adequate data. Consequently, it fails to capture the essential patterns necessary for accurate predictions. Underfitting may also stem from an ill-suited model choice. This issue can often be mitigated by selecting a different model, allocating more time for training, or augmenting the training dataset.

To address this problem, we can increase the model complexity by adding layers, using a deeper neural network.

5.4 OVERFITTING

While underfitting is relatively straightforward to understand, overfitting[26] is often less intuitive. It occurs when the model becomes excessively accurate, essentially memorizing patterns present only in the training data. Consequently, its performance on new data significantly diminishes. Overfitting can be identified by comparing error rates between training and validation datasets. When the model performs exceptionally well on the training data but poorly on the validation data, it indicates an overfitting issue.

Some recommended solutions include employing data augmentation or applying regularization techniques such as L_1,L_2 regularization or dropout to decrease model complexity.

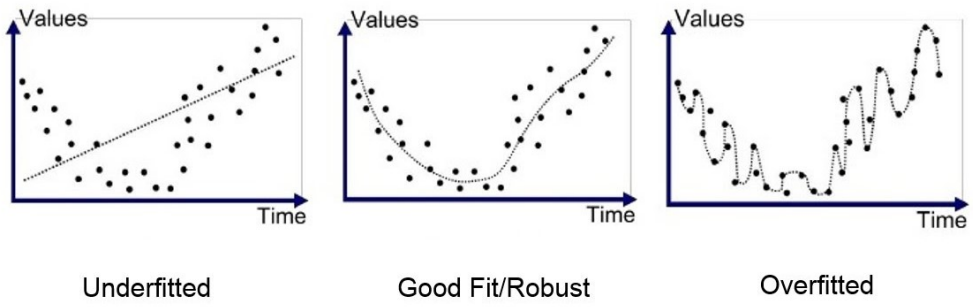

Figure 1.7: Overfitting vs Underfitting

6 CONCLUSION

The training of deep learning models often demands extensive computational resources and time due to the complexity of their architectures and the scale of datasets involved. Additionally, challenges such as vanishing and exploding gradients, overfitting, and underfitting can impede the optimization process. Addressing these challenges requires the application of regularization techniques, careful data preprocessing, and thoughtful model architecture design. Despite these obstacles, optimizing deep learning models is pivotal for achieving state-of-the-art performance in diverse domains, driving advancements in artificial intelligence and facilitating the development of innovative applications.

In this chapter, we focus on several models and discuss various optimization methods used for training deep learning models. Finally, we delve into common issues that arise during the training process.

CHAPTER 2

ADAM-BASED OPTIMIZERS

1 INTRODUCTION

Adam is a widely popular optimization algorithm in deep learning due to its effectiveness and ease of use. However, many variants have been proposed to improve its generalization and address its drawbacks. In this chapter, we detail the Adam algorithm, outline some of its limitations, and then present several of its variants, categorized based on their modifications or the specific issues they address.

2 ADAPTIVE MOMENT ESTIMATION (ADAM)

Adam [6, 18] is an algorithm that combine AdaGrad optimizer which performs effectively with sparse gradients. With RMSProp optimizer which is suitable for both online and non-stationary settings. First of all, it computes the partial derivative (gradient) of the objective function f w.r.t its parameters θ, simply because the gradient indicates the impact of changing the parameter θ on the function f.

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \tag{2.1}$$

Then, using the exponential moving averages method, it computes the first and the second moments of the gradient m_t and v_t which represent the mean and uncenter variance of the gradient direction respectively.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.2}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.3}$$

Simply because the first moment provide more precise directional information, making the second moment a superior choice for scaling learning rates. And since the initialization of moving averages as 0's pushes moment estimations towards zero, especially during the initial time-steps, it computes bias-corrected versions $\hat{m}_t$ and $\hat{v}_t$.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.4}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.5}$$

Finally, we update the parameters using the following update rule:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.6}$$

with α is learning rate, and ϵ is just a small constant that is added in the denominator for numerical stability.

Adam uses smoothing parameters β_1, β_2 to reduce the variance of parameter updates, thereby improving training stability and convergence.

β_1 : This parameter controls the exponential decay rate for the first moment estimate m_t of the gradients. A typical value for β_1 is 0.9. By using a value less than 1, the optimizer gives more weight to recent gradients, helping to adapt to changes in the gradient direction.

β_2 : This parameter controls the exponential decay rate for the second moment estimate v_t of the gradients. A typical value for β_2 is 0.999. Similar to β_1 , using a value less than 1 gives more weight to recent squared gradients, which helps to adaptively adjust the learning rates for each parameter.

Algorithm 2 Adam Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$
end while

3 ADAM'S LIMITATIONS

While the Adam optimizer is commonly used and effective in various deep learning applications, it is not without its own challenges and potential drawbacks. Recognizing these obstacles is essential for making well-informed decisions when selecting an optimization algorithm. The following presents an outline of certain issues associated with Adam:

3.1 MEMORY REQUIREMENTS

Adam requires more memory compared to simpler optimizers like SGD because it maintains two moving averages per parameter. This additional memory usage is attributed to storing both the exponentially decaying average of past gradients (first moment) and the exponentially decaying average of past squared gradients (second moment)[33].

3.2 POOR GENERALIZATION

Generalization denotes how well a solution θ performs across a wider population. For example, in classification tasks, generalization is usually defined by the classification error instead of the cross-entropy. And the generalization bound(gap) represents the difference between the error on a training set and the error on an unseen data point for a model [14, 44].

It has been observed that Adam can sometimes converge to a solution with a worse test error in many deep learning tasks, rendering it no better than a random guess. This

issue arises due to Adam's behavior during the training process. In the early stages, Adam is more likely to fit the noise in the data. As training progresses, it retains the learned patterns and tends to converge at a local stationary point. Consequently, Adam achieves an excellent training error but performs poorly in terms of generalization on the test set[51]. This issue can be caused by the following limitations of Adam:

3.3 REGULARIZATION

There are two regularization techniques that we use to addressing the overfitting issues in machine learning:

Weight Decay Regularization: applies a decay factor $(1 - \lambda)$ to the parameters at the previous step t

$$\theta_t = (1 - \lambda)\theta_{t-1} - \alpha\nabla_\theta f(\theta_{t-1}) \tag{2.7}$$

L_2 **Regularization**: defines a loss term which is:

$$L(\theta_t) = f(\theta_t) + \frac{\lambda}{2}||\theta_t||_2^2 \tag{2.8}$$

so we have:

$$\nabla_\theta L(\theta_t) = \nabla_\theta f(\theta_t) + \lambda\theta_t \tag{2.9}$$

However, under SGD optimization, L_2 regularization is equivalent to re-parameterized weight decay regularization. We can see this equivalence if we rewrite the L_2 regularization as bellow:

$$\theta_t = \theta_{t-1} - \alpha(\nabla_\theta f(\theta_{t-1}) + \lambda\theta_{t-1}) \tag{2.10}$$

Due to this equivalence, most deep learning frameworks typically only implement L_2 regularization. However, in the context of Adam optimization, these two techniques are not equivalent. Utilizing L_2 regularization can result in weights with substantial gradients being less regulated compared to when using weight decay [22, 23].

3.4 EXPONENTIAL AVERAGING

In Adam, the moving average of the squared gradients is used to update the learning rate for each parameter. However, in such scenarios, it's commonly noted that certain mini-

batches provide significant gradients, but not very often. Although these gradients offer valuable information, their impact in learning rate diminishes rapidly due to exponential averaging (second moment) leading to poor convergence[30].

3.5 THE DIRECTION MISSING PROBLEM

The issue known as the "direction missing problem"[44] in Adam describes a scenario wherein the vector for parameter updates loses correlation with the descent direction across the training loss landscape. This can potentially result in divergence during the training of large language models. In other words, Adam adapts the global learning rate to each scalar parameter independently, such that the gradient of each parameter is normalized by a running average of its magnitudes, which changes the direction of the gradient.

3.6 THE ILL-CONDITIONING PROBLEM:

Ill-conditioning [13] refers to situations where small changes in input parameters lead to significant changes in the output, due to interactions with other parameters, such as the scaling factors in batch normalization. This sensitivity can cause instability in the training process, leading to issues like slow convergence or divergence of the model. Therefore, when considering different magnitudes of an input weight vector, the updates provided by Adam may produce differing impacts on the overall network function.

3.7 SIGN DESCENT

In scenarios with low variance, Adam uses the "Sign Descent" approach instead of the "gradient descent" approach which lead to a deviation from the true gradient direction.[1]

Given the following assumptions:

- presuming g_t is drawn from a stationary distribution, hence after bias correction we have $E(v_t) = E(g_t)^2 + Var(g_t)$
- low-noise assumption, assume $E(g_t)^2 \gg Var(g_t)$
- Assume β_1^t is small. Then:

$$\Delta\theta_t^{Adam} = -\alpha\frac{m_t}{\sqrt{v_t}+\epsilon} \approx -\alpha\frac{E(g_t)}{\sqrt{E(g_t)^2+Var(g_t)}+\epsilon} \approx -\alpha\frac{E(g_t)}{||E(g_t)||} \approx -\alpha Sign(E(g_t)) \tag{2.11}$$

3.8 THE SIGN OF GRADIENT

In the Adam optimization algorithm, the update direction for a weight is dictated by the sign of the stochastic gradients. In simpler terms, if the gradient is positive, the weight will be adjusted in one direction, and if it's negative, it will be adjusted in the opposite direction. However, in certain cases, when the optimizer oscillates in the one direction and continually increases in other one, Adam only uses the magnitude of the gradient and ignore its sign [50].

3.9 INFLUENCE OF NOISE

Adam can be sensitive to the presence of outliers or extreme values in the gradients. Outliers may disproportionately affect the calculation of the moving averages, leading to suboptimal updates and impacting the overall performance of the optimizer[17].

These limitations impact the performance of the Adam optimizer, leading to the development of several variants aimed at addressing these issues.

4 ADAM'S VARIANTS

Researchers have developed several variants of Adam to address specific challenges or improve its performance in various scenarios. These variants often aim to mitigate issues like convergence speed, robustness to noise, or generalization capabilities.
We classify these versions according to the changes made or the strategies employed to address the same problem. As illustrated in the figure2.1.

4.1 LEARNING RATE MODIFIERS

Adam may results in extreme learning rates which lead to convergence issues and unstable training. In this category we introduce Adam's variants that reduce the variance of the

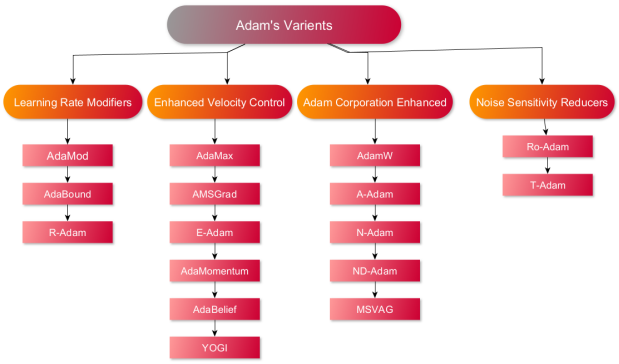

Figure 2.1: Adam's Variants

adaptive step-sizes in different manner.

4.1.1 ADAMOD OPTIMIZER

The idea behind AdaMod [7] (Adaptive and Momental Bound) is to restrict the adaptive learning rates with adaptive and momental upper bound in order to smoothing out the unexpected large learning rates which make AdaMod more stable during the training process. AdaMod introduces two key modifications:

Applying the exponentail moving average to the adaptive learning rates computed by Adam to get smoothed learning rates using the following operation:

$$s_t = \beta_3 s_{t-1} + (1 - \beta_3)\eta_t \tag{2.12}$$

Using the smoothed learning rates as an adaptive upper bound on the original learning rates to eliminate extremely large values using:

$$\hat{\eta}_t = min(\eta_t, s_t) \tag{2.13}$$

This "Long-Term Memory" of past gradients in the form of smoothed learning rates is intended to improve the stability of the training process and get better generalization performance.

Algorithm 3 AdaMod Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$,$s_t \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
$\eta_t \leftarrow \frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon}$
$s_t \leftarrow \beta_3 s_{t-1} + (1 - \beta_3)\eta_t$
$\hat{\eta}_t \leftarrow min(\eta_t, s_t)$
$\theta_t gets \theta_{t-1} - \hat{\eta}_t \hat{m}_t$
end while

with $\eta_t, \hat{\eta}_t$ are the learning rates and s_t is the moving average of the learning rates.

4.1.2 ADABOUND OPTIMIZER

AdaBound [24] tries to clip the learning rates dynamically to keep them within a desired range. Using the same estimates m_t, v_t as in Adam, AdaBound computes the initial learning rate using this equation :

$$\eta'_t = \frac{\alpha}{\sqrt{v_t}} \tag{2.14}$$

Then, AdaBound will clip this adjusted learning rate to ensure that it's within the lower η_l and the upper η_u bounds, which can be a constant or a function of step t.

$$\hat{\eta}_t = Clip(\eta'_t, \eta_l, \eta_u) \tag{2.15}$$

After obtaining the bounded learning rate η'_t, we scale it using

$$\eta_t = \frac{\hat{\eta}_t}{\sqrt{t}} \tag{2.16}$$

Then we use it in the updating rule:

$$\theta_t = \theta_{t-1} - \eta_t m_t \tag{2.17}$$

Algorithm 4 AdaBound Optimizer

$m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
$\eta'_t \leftarrow \frac{\alpha}{\sqrt{v_t}}$
$\hat{\eta}_t \leftarrow Clip(\eta'_t, \eta_l, \eta_u)$
$\eta_t \leftarrow \frac{\hat{\eta}_t}{\sqrt{t}}$
$\theta_t \leftarrow \theta_{t-1} - \eta_t m_t$
end while

4.1.3 R-ADAM OPTIMIZER

R-Adam[20] introduces a rectification mechanism that dynamically adjusts the learning rate, making it more robust and less sensitive to the choice of hyper-parameters. The main idea behind R-Adam is to address the variance of the adaptive learning rate at the beginning of the training.

Early in training, the variance of the adaptive learning rates can be very high, leading to unstable and inefficient training. R-Adam introduces a term called the "variance rectification term" based on the ratio of the magnitude of the current gradient to the magnitude of the gradients up to that point, which helps stabilize the training in the initial phases.

If the variance is tractable, R-Adam calculates the rectifier term using this equation:

$$r_t = \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}} \tag{2.18}$$

with

$$\rho_t = \rho_\infty - 2t\frac{\beta_2^t}{1 - \beta_2^t} \tag{2.19}$$

$$\rho_\infty = \frac{2}{1 - \beta_2} - 1 \tag{2.20}$$

Then,it reduces the learning rate to avoid overshooting the minimum using:

$$l_t = \sqrt{\frac{1-\beta_2^t}{v_t}} \tag{2.21}$$

The updating rule will be like:

$$\theta_t = \theta_{t-1} - \alpha r_t \hat{m}_t l_t \tag{2.22}$$

For the other case(the variance isn't large), the only thing that has changed between Adam and R-Adam is the updating rule formula

$$\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t \tag{2.23}$$

Algorithm 5 R-Adam Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
  $\rho_\infty \leftarrow \frac{2}{1-\beta_2} - 1$
  $\rho_t \leftarrow \rho_\infty - 2t\frac{\beta_2^t}{1-\beta_2^t}$
  if the variance is large i.e. $\rho_t > 4$ **then**
    $l_t \leftarrow \sqrt{\frac{1-\beta_2^t}{v_t}}$
    $r_t \leftarrow \sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$
    $\theta_t \leftarrow \theta_{t-1} - \alpha r_t \hat{m}_t l_t$
  else
    $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t$
  end if
end while

4.2 ENHANCED VELOCITY CONTROL

The velocity term (second moment v_t) in Adam plays a crucial role in determining the magnitude of parameter updates (step-size). It integrates both the current gradient and the mo-

mentum from previous steps, fostering smoother and more precise adjustments throughout the optimization process. Several variants of the Adam optimizer have been developed to modify the calculation of the second moment for achieving more controlled adjustments during training. Then improve the performance and stability of the optimization process. below we enumerate some of these variants.

4.2.1 ADAMAX OPTIMIZER

AdaMax [6] is a variant of Adam based on the infinity norm. It updates the exponentially weighted infinity norm which make it simply and stable than Adam.
AdaMax uses the estimate of the first moment m_t the same way as Adam, but instead of the second moment v_t it uses the infinity norm of the gradients.
The second moment v_t is the L_2 norm of the gradients and it can be generalized to L_p norm where $p > 2$, so, in this case it would represent the running average of the L_p norm of the gradients and it's computed using :

$$v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p)|g_t|^p \tag{2.24}$$

With this generalization formulation for v_t, AdaMax lets $p = \infty$ which results in very simple and stable optimizer. So, it uses u_t instead of v_t which is computed using :

$$u_t = max(\beta_2 u_{t-1}, |g_t|) \tag{2.25}$$

After that, it applies the bias correction only to the first moment $\hat{m}$ then the update rule is very simple:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{u_t} \tag{2.26}$$

Algorithm 6 AdaMax Optimizer

Set $m_0 \leftarrow 0$, $u_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $u_t \leftarrow max(\beta_2 u_{t-1}, |g_t|)$
  $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
  $\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{u_t}$
end while

4.2.2 AMSGRAD OPTIMIZER

AMSGrad [30] is derived from "Adaptive Moment Estimation with a Stable Gradient". It uses the same estimate of the first m_t and the second v_t moments, but instead of Adam's bias correction it simply takes the maximum of all previous and current estimates of the second moment

$$\hat{v}_t = max(\hat{v}_{t-1}, v_t) \tag{2.27}$$

To have non-increasing step-size avoiding the exponential averaging problem of Adam. The updating rule is similar to Adam which is:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{v_t} + \epsilon} \tag{2.28}$$

Algorithm 7 AMSGrad Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
  $\hat{v}_t \leftarrow max(\hat{v_{t-1}}, v_t)$
  $\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while

4.2.3 E-ADAM OPTIMIZER

E-Adam [46] focuses on the impact of the constant ϵ. It provides a simple modification without requiring addition hyper-parameters or computational costs.
The constant ϵ in E-Adam is added directly to the second moment v_t before the bias corrected $\hat{v}_t$ as in the original Adam. This modification allows E-Adam to take smaller steps when the parameters θ are close to the global minimum, which can help avoid overshooting. Additionally, E-Adam incorporates an adaptive ϵ rather than a fixed constant as in Adam.

Algorithm 8 E-Adam Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 + \epsilon$
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$
end while

4.2.4 ADAMOMENTUM OPTIMIZER

Expanding on the Adam optimizer, AdaMomentum [43] replaces the gradient within the velocity term v_t with its momentum m_t, thereby enhancing the smoothing of the exponentially moving average (EMA), and changing the location of ϵ like in E-Adam optimizer. This alteration can enhance training by offering better control over step sizes, thereby mitigating the issue of overshooting.

Algorithm 9 AdaMomentum Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)\ m_t^2 + \epsilon$
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$
end while

4.2.5 ADABELIEF OPTIMIZER

AdaBelief [50] tries to reduce the generalization gap between the SGD and the adaptive methods by dynamically adjusting the step-size in Adam, based on the "belief" correction between the actual gradient g_t and the predicted gradient m_t. When g_t significantly diverges from m_t, indicating a tentative belief in g_t, the optimizer proceeds cautiously with a small step. However, when g_t closely aligns with m_t, signaling strong belief in g_t, it confidently takes larger steps.

Algorithm 10 AdaBelief Optimizer

Set $m_0 \leftarrow 0$, $s_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$s_t \leftarrow \beta_2 s_{t-1} + (1 - \beta_2)(g_t - m_t)^2$
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
$\hat{s}_t \leftarrow \frac{s_t}{1-\beta_2^t}$
$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{s}_t + \epsilon}}$
end while

4.2.6 YOGI OPTIMIZER

YOGI [47] is proposed to address the convergence issues especially in non-convex stochastic optimization problems. It uses the same estimate m_t as in Adam, but for the second

moment v_t YOGI adds a factor which is $sign(v_{t-1} - g_t^2)$ and the rest (bias correction, update rule) are the same.

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) sign(v_{t-1} - g_t^2) g_t^2 \tag{2.29}$$

to understand how this modification affects the estimate of v_t, we need to take look at the change in $v_t - v_{t-1}$ vector and see how they are different between Adam and YOGI.

Table 2.1: The different between Adam and YOGI.

Optimizer	Adam	YOGI
$v_t - v_{t-1}$	$-(1 - \beta_2)(v_{t-1} - g_t^2)$	$-(1 - \beta_2) Sign(v_{t-1} - g_t^2) g_t^2$

When $v_{t-1} - g_t^2 < 0$, it results in increasing v_t in both Adam and YOGI. However, in Adam the magnitude of the change is depends on $v_{t-1} - g_t^2$, but in YOGI, it's only depends on g_t^2 resulting in controlled increase/ decrease of the learning rates.

Algorithm 11 YOGI Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)\ sign(v_{t-1} - g_t^2)\ g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$
  $\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t}$
  $\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while

4.3 ADAM CORPORATION ENHANCED

Below, we provide a list of some modified versions of Adam that aim to enhance its performance by introducing additional terms.

4.3.1 ADAMW OPTIMIZER

When we are using adaptive methods like Adam for training, it's best to use weight decay regularization instead of L_2 regularization. That's where AdamW [21] comes in to modify the updating rule for incorporating weight decay regularization in Adam.
Most of the steps such as the estimates m_t, v_t and applying bias correction are the same as in Adam. The only difference is in the update equation where we will add the regularization term $\lambda\theta_t$ and it will be :

$$\theta_t = \theta_{t-1} - \alpha\frac{\hat{m}_t}{\sqrt{v_t}+\epsilon} + \lambda\theta_{t-1} \tag{2.30}$$

Algorithm 12 AdamW Optimizer

$m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t+1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)g_t^2$
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
$\theta_t = \theta_{t-1} - \alpha\frac{\hat{m}_t}{\sqrt{v_t}+\epsilon} + \lambda\theta_{t-1}$
end while

4.3.2 A-ADAM OPTIMIZER

The Accelerate Adam (A-Adam) [38] algorithm which build on the Adam optimizer by keeping track of the exponentially decaying average of the past updates .
In addition to the first m_t and the second v_t moments of the gradients as in Adam, the current update rule incorporates a small value d proportional to the sign of the previous updates which is meant to accelerate the process along dimensions where the gradients consistently point in the same direction.

Algorithm 13 A-Adam Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
$d \leftarrow \Delta\theta_{t-1} * sing(g_t)(1 - \beta_1)$
$\theta_t = \theta_{t-1} - (\alpha \frac{\beta_1 \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + d)$
end while

4.3.3 N-ADAM OPTIMIZER

N-Adam [9] aims to improve the speed of convergence and the quality of the learned models by modifying Adam's momentum component with Nesterov accelerated gradient (NAG). It proves to be a favorable optimizer for training models with sparse gradients. With N-Adam, when performing the bias correction for the first moment m_t, we also include the current gradients g_t as in the formula:

$$\hat{m}_t = \frac{\beta_1}{1 - \beta_1^{t+1}} m_t + \frac{(1 - \beta_1)}{1 - \beta_1^t} g_t \tag{2.31}$$

Algorithm 14 N-Adam Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
$\hat{m}_t \leftarrow \frac{\beta_1}{1-\beta_1^{t+1}} m_t + \frac{(1-\beta_1)}{1-\beta_1^t} g_t$
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while

4.3.4 ND-ADAM OPTIMIZER

ND-Adam[49] It fixes the direction missing problem by adapting the learning rate to each weight vector rather than to each individual weight. This is achieved by making the learning rate a linear combination of historical gradients. Specifically, the bias-corrected first moment $\hat{m}_t$ of Adam is generalized from a scalar to a vector:

$$\hat{m}_t(w_i) = \frac{m_t(w_i)}{1 - \beta_1^t} \tag{2.32}$$

This approach ensures that the learning rate adjustment considers the entire weight vector's history, thereby maintaining the correct update direction.

It addresses the ill-conditioning problem by explicitly normalizing each weight vector:

$$w_{i,t} = \frac{\hat{w_{i,t}}}{||\hat{w_{i,t}}||_2} \tag{2.33}$$

Additionally, it preserves only the gradient component that is orthogonal to $w_{i,t-1}$ using:

$$g_t(w_i) = \hat{g}_t(w_i) - (\hat{g}_t(w_i).w_{i,t-1})w_{i,t-1} \tag{2.34}$$

This optimization focuses solely on the direction of the weight vector, allowing for precise control of the effective learning rate.

ND-Adam divides the trainable parameters θ, into two sets, θ^v and θ^s, such that $\theta^v = \{w_i \mid i \in \mathbb{N}\}$ which includes the weight vectors, and $\theta^s = \{\theta|\theta^v\}$ which includes all other parameters. The θ^v set is updated using the previously described rules, while the θ^s set is updated using the standard Adam rules. The learning rates for the two sets of parameters are denoted by α_t^v and α_t^s respectively.

Algorithm 15 ND-Adam Optimizer

Set $t \leftarrow 0$
for $i \in \mathbb{N}$ **do**
  $w_{i,0} \leftarrow \frac{w_{i,0}}{||w_{i,0}||_2}$
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
end for
while $t < T$ **do**
  $t \leftarrow t + 1$
  for $i \in \mathbb{N}$ **do**
    $\hat{g}_t(w_i) \leftarrow \nabla_\theta f_t(w_i)$
    $g_t(w_i) \leftarrow \hat{g}_t(w_i) - (\hat{g}_t(w_i).w_{i,t-1})w_{i,t-1}$
    $m_t(w_i) \leftarrow \beta_1 m_{t-1}(w_i) + (1-\beta_1)g_t(w_i)$
    $v_t(w_i) \leftarrow \beta_2 v_{t-1}(w_i) + (1-\beta_2)||g_t(w_i)||_2^2$
    $\hat{m}_t(w_i) \leftarrow \frac{m_t(w_i)}{1-\beta_1^t}$
    $\hat{v}_t(w_i) \leftarrow \frac{v_t(w_i)}{1-\beta_2^t}$
    $\hat{w_{i,t}} \leftarrow w_{i,t-1} - \alpha_t^v \frac{\hat{m_t}(w_i)}{\sqrt{\hat{v_t}(w_i)+\epsilon}}$
    $w_{i,t} \leftarrow \frac{\hat{w_{i,t}}}{||\hat{w_{i,t}}||_2}$
  end for
  $\theta_t^s \leftarrow AdamUpdate(\theta_{t-1}^s, \alpha_t^s)$
end while

4.3.5 MSVAG OPTIMIZER

The Momentum Sign Variance Adapted Gradient (MSVAG)[1] combines momentum-based optimization with adaptive learning rate adjustment based on the sing variance of the gradient. The main idea is to monitor the gradients during train process, because the sign variance reflects how much the gradients are fluctuating in different directions. e.i high sign variance indicates that the gradients are changing rapidly and erratically, while low sign variance suggests a more stable gradient direction.

Like Adam, MSVAG compute the first m_t and the second v_t moments and their bias correction versions $\hat{m}_t$ and $\hat{v}_t$. However, it adds

$$\rho(\beta, t) := \frac{(1-\beta)(1+\beta^{t+1})}{(1+\beta)(1-\beta^{t+1})} \tag{2.35}$$

Then use it to calculate :

$$s_t = \frac{1}{1-\rho(\beta,t)}(v_t - \hat{m}_t{}^2) \tag{2.36}$$

$$\gamma = \frac{\hat{m}_t{}^2}{\hat{m}_t{}^2 + \rho(\beta,t)s_t} \tag{2.37}$$

The update rule of MSVAG is:

$$\theta_t = \theta_{t-1} - \alpha(\gamma \cdot \hat{m}_t) \tag{2.38}$$

Algorithm 16 MSVAG Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t+1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta m_{t-1} + (1-\beta)g_t$
$v_t \leftarrow \beta v_{t-1} + (1-\beta)g_t^2$
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta^t}$
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta^t}$
$\rho(\beta,t) \leftarrow \frac{(1-\beta)(1+\beta^{t+1})}{(1+\beta)(1-\beta^{t+1})}$
$s_t \leftarrow \frac{1}{1-\rho(\beta,t)}(v_t - \hat{m}_t{}^2)$
$\gamma \leftarrow \frac{\hat{m}_t{}^2}{\hat{m}_t{}^2+\rho(\beta,t)s_t}$
$\theta_t \leftarrow \theta_{t-1} - \alpha\ (\gamma \cdot \hat{m}_t)$
end while

MSVAG exposes only two hyper-parameters, α and β.

4.4 NOISE SENSITIVITY REDUCERS

In environments with high levels of noise, where gradients fluctuate widely due to random perturbations, the standard Adam optimizer's performance may suffer. The presence of noise can result in imprecise gradient estimates, thereby disrupting the optimization process. Here are some versions of Adam specifically designed to address this issue.

4.4.1 RO-ADAM OPTIMIZER

Robust Adam [45] is online gradient learning method for time series prediction that modifies Adam to detect the outliers and adaptively tunes the learning rate using a relative

prediction error term r_t which indicates whether the point is an outlier or not. A higher value of r_t indicates that the current point is more likely to be an outlier. We calculate this term using:

$$r_t = \begin{cases} \min\left\{\max\left\{k, \left\|\frac{f(\theta_{t-1})}{f(\theta_{t-2})}\right\|\right\}, K\right\} & if||f(\theta_{t-1})|| > ||f(\theta_{t-2})|| \\ \min\left\{\max\left\{\frac{1}{K}, \left\|\frac{f(\theta_{t-1})}{f(\theta_{t-2})}\right\|\right\}, \frac{1}{k}\right\} & otherwise \end{cases} \tag{2.39}$$

Where k and K are the lower and upper thresholds. We then compute the moving average of r_t to obtain a smoother estimation.

$$d_t = \beta_3 d_{t-1} + (1 - \beta_3) r_t \tag{2.40}$$

Adding the estimation d_t to the updating rule allows to control the adaptive learning rate

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{d_t \sqrt{\hat{v}_t} + \epsilon} \tag{2.41}$$

Algorithm 17 Ro-Adam Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$,$d_0 \leftarrow 1$, $f(\theta_0) = f(\theta_{-1}) \leftarrow 1$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta m_{t-1} + (1 - \beta) g_t$
  $v_t \leftarrow \beta v_{t-1} + (1 - \beta) g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{1-\beta^t}$
  $\hat{v}_t \leftarrow \frac{v_t}{1-\beta^t}$
  if $||f(\theta_{t-1})|| > ||f(\theta_{t-2})||$ **then**
    $r_t \leftarrow \min\left\{\max\left\{k, \left\|\frac{f(\theta_{t-1})}{f(\theta_{t-2})}\right\|\right\}, K\right\}$
  else
    $r_t = \min\left\{\max\left\{\frac{1}{K}, \left\|\frac{f(\theta_{t-1})}{f(\theta_{t-2})}\right\|\right\}, \frac{1}{k}\right\}$
  end if
  $d_t \leftarrow \beta_3 d_{t-1} + (1 - \beta_3) r_t$
  $\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{d_t \sqrt{\hat{v}_t} + \epsilon}$
end while

4.4.2 T-ADAM OPTIMIZER

The moving average estimators m_t and v_t in Adam make the gradients come from the Gaussian distribution which is sensitive to outliers and noise. That's why Adam's performance tends to degrade when trained on a dataset with a high level of noise.
T-Adam[17] is a variant of Adam that substitutes the Gaussian distribution with the Student T distribution, known for its robust properties in probability distributions. The key distinction between Adam and T-Adam lies in their computation of the first moment, which T-Adam calculates using the following formula:

$$m_t = \frac{W_{t-1}}{W_{t-1} + w_t} m_{t-1} + \frac{w_t}{W_{t-1} + w_t} g_t \tag{2.42}$$

where:

$$w_t = (\nu + d) \left(\nu + \sum_j \frac{(g_j^t - m_j^{t-1})^2}{v_{t-1} + \epsilon} \right)^{-1} \tag{2.43}$$

And

$$W_t = \frac{2\beta_1 - 1}{\beta_1} W_{t-1} + w_t \tag{2.44}$$

T-Adam require two additional hyper-parameters, the degrees of freedom ν which control the robustness, and d is the dimension of the gradients g_t.

Algorithm 18 T-Adam Optimizer

Set $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
$W_0 \leftarrow \frac{\beta_1}{1-\beta_1}$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $w_t \leftarrow (\nu + d) \left(\nu + \sum_j \frac{(g_j^t - m_j^{t-1})^2}{v_{t-1} + \epsilon} \right)^{-1}$
  $m_t \leftarrow \frac{W_{t-1}}{W_{t-1} + w_t} m_{t-1} + \frac{w_t}{W_{t-1} + w_t} g_t$
  $W_t \leftarrow \frac{2\beta_1 - 1}{\beta_1} W_{t-1} + w_t$
  $v_t \leftarrow \beta v_{t-1} + (1 - \beta) g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta^t}$
  $\hat{v}_t \leftarrow \frac{v_t}{1 - \beta^t}$
  $\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while

5 CONCLUSION

To summarize, Adam and its variations are powerful optimization algorithms that have significantly contributed to the success of deep learning. Their flexibility and effectiveness make them popular choices for neural network training. In this chapter, we detail the Adam algorithm, outline some of its limitations, and then present several of its variants, categorized based on their modifications or the specific issues they address.

CHAPTER 3

METHODOLOGY OF COMPARATIVE STUDY

1 INTRODUCTION

In this study, we use various Adam variants to train different models in order to identify the best-performing optimizer among the following 15 options: Adam, AdaMod, AdaBound, R-Adam, AdaBelief, AdaMax, AMSGrad, E-Adam, YOGI, AdamW, N-Adam, ND-Adam, MSVAG, T-Adam, and Ro-Adam.

This chapter details the experimental setup requirements for a comparative study across three tasks: image classification, language modeling, and time series forecasting. It covers the datasets and their preprocessing methods, the training configurations, and the evaluation metrics utilized

2 IMAGE CLASSIFICATION TASK

Image classification represents a core task in computer vision, aiming to categorize images into predefined classes or categories. In this study, we employ the following configuration.

2.1 DATASETS

MNIST Dataset: It[1] is one of the most well-known benchmark in computer vision. It contains 60,000 training images and 10,000 testing images, each of which is a grayscale image of size 28x28 pixels. Each image corresponds to a handwritten digit from 0 to 9.
We use this dataset because it exhibits sparse characteristics, with each image typically having a black background and the digit occupying only a small portion of it.

CIFAR10 Dataset: It[2] is another widely used benchmark. It contains 60,000 32x32 color images in 10 different classes, with 6,000 images per class. These classes include common objects such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.
CIFAR10 has dense characteristics, making it more complex. This increased complexity helps in enhancing model performance and robustness.

2.2 PREPROCESSING

We apply identical preprocessing steps to both datasets by scaling the data to a range of 0 to 1 and dividing them into 32 batches.

2.3 MODEL'S ARCHITECTURE

Two models are trained. The first model is a simple CNN with 5 layers, as illustrated in the figure 3.1, and trained on the MNIST dataset. The idea behind using this model is to demonstrate that Adam's limitations are primarily observed in deeper models. Employing a straightforward model allows us to assess the modifications' effects without being influenced by these limitations.

The second model is a ResNet34 architecture, trained on the CIFAR10 dataset. ResNet addresses the vanishing/exploding gradient problem during training by leveraging two fundamental properties. Firstly, each block in the network augments the data, streamlining

[1] https://pytorch.org/vision/main/generated/torchvision.datasets.MNIST.html
[2] https://pytorch.org/vision/main/generated/torchvision.datasets.CIFAR10.htmltorchvision.datasets.CIFAR10

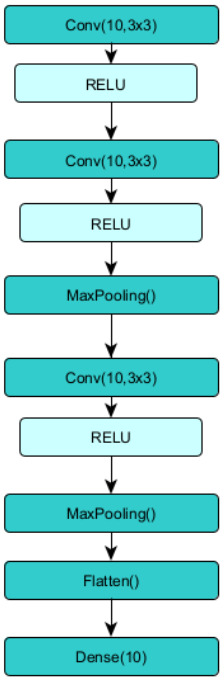

Figure 3.1: simple CNN

learning and improving information preservation. Secondly, ResNet incorporates skip connections, leading to shorter gradient paths, thereby facilitating more seamless and efficient training.

2.4 LOSS FUNCTIONS:

Cross Entropy Loss: Also known as log loss, is a commonly used loss function for classification tasks, including multi-class classification. It measures the difference between the predicted probability distribution and the actual probability distribution of the classes. its formula is:

$$\text{CrossEntropyLoss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log(p_{i,j})$$

such as N is the number of examples in the batch.

C is the number of classes.

$y_{i,j}$ is a binary indicator of whether class j is the correct classification for example i

$p_{i,j}$ is the predicted probability that example i belongs to class j.

2.5 EVALUATION METRICS

Accuracy score: It refers to a metric used to evaluate the performance of a classification model. It measures the percentage of predictions made by the model that match the actual labels in the dataset. It's computed using

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \times 100\%$$

High accuracy scores indicate that the model is making correct predictions most of the time.

Precision: It represents the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives and false positives). It's calculated using the formula:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

High precision indicates that the model is making accurate positive predictions and has a low rate of falsely classifying negative instances as positive.

Recall: Also known as sensitivity or true positive rate. It measures the ability of the model to correctly identify positive instances out of all actual positive instances. It is calculated using the formula:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

High recall indicates that the model is effectively capturing a large percentage of actual positive instances, minimizing the number of false negatives (instances incorrectly classified as negative).

F_1 **score:** It is the harmonic mean of precision and recall. It provides a balance between precision (the ability of the classifier not to label a negative sample as positive) and recall (the ability of the classifier to find all the positive samples). The formula for calculating the F_1 score is:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

High F_1 scores indicate that the model has both high precision and high recall, meaning it is making accurate positive predictions while also capturing a high percentage of actual positive instances

Matthews Correlation Coefficient: It is a measure used in machine learning to evaluate the quality of binary classification predictions, particularly in imbalanced datasets. Its ranges from $[-1, 1]$, where 1 indicates perfect prediction, 0 indicates random prediction, and -1 indicates total disagreement between prediction and observation. We can compute the MCC score using this formula:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

3 LANGUAGE MODELING TASK

Language modeling, an essential task in natural language processing (NLP), revolves around predicting the next word or sequence of words within a given context. Its principal objective is to grasp the structure and patterns of a language, enabling the generation of meaningful text. In our work, we use the following setting for this task.

3.1 DATASET

Penn Treebank dataset: It[3] is a widely used corpus in natural language processing (NLP) and computational linguistics. It consists of a large collection of English text derived from various sources, including newspaper articles, transcripts of spoken language, and literature. It is known for its manually annotated syntactic structures, particularly parse trees, which represent the grammatical structure of sentences.

[3] https://pytorch.org/text/stable/datasets.html?highlight=penn+treebanktorchtext.datasets.PennTreebank

3.2 PREPROCESSING

We segment it into sentences and further divide these into 128 batches. The target is the last word of each sentence, while the input consists of the remaining words.

3.3 MODEL'S ARCHITECTURE

We train an autosentence completion model using an LSTM architecture. This model consists of an embedding layer, followed by three LSTM layers, each with 128 nodes, and a final dense layer for generating the output.

3.4 LOSS FUNCTIONS

Categorical Cross Entropy Loss It's a specific adaptation of cross-entropy loss applied to multi-class classification tasks, particularly when labels are encoded in a one-hot format. Widely employed in language modeling, it's valued for its efficacy in quantifying the disparity between predicted probability distributions and actual ground truths.

$$H(p, q) = \sum_{i=1}^{n} p(x_i) \log q(x_i)$$

Where $q(x_i)$ and $p(x_i)$ represent the probability distributions of predictions and targets respectively and x_i is an input vector.

3.5 EVALUATION METRICS

perplexity: It[25] measures how well a probability distribution or model predicts a sample. In the context of language modeling, it quantifies how well a language model predicts a sequence of tokens (e.g., words or characters). A lower perplexity indicates better performance, as it means the model assigns higher probabilities to the actual tokens in the sequence.

Given a sequence of tokens $w_1, w_2, ..., w_n$, the perplexity is calculated as

$$PP = \sqrt[n]{\frac{1}{\prod_{i=1}^{n} P(w_i|w_1, w_2, ..., w_{i-1})}}$$

or

$$PP = Exp(CrossEntropyLoss)$$

In addition to the matrix below, we also utilize the **accuracy** and F_1 **score** that were mentioned earlier.

4 TIME SERIES FORECASTING TASK

Time series forecasting is a branch of statistical analysis and machine learning focused on predicting future values based on historical data points that are ordered chronologically. It's used across various domains like finance, economics, weather forecasting, and more. For this task, we use the following configurations.

4.1 DATASET

Amazon Stock Dataset: It[4] refers to a collection of historical data related to the trading activity of Amazon.com Inc. (ticker symbol: AMZN) on stock exchanges. Such datasets usually include information such as the opening price, closing price, highest price, lowest price, trading volume, and adjusted closing price of Amazon stock for each trading day.

4.2 PREPROCESSING

We generate a time series dataset comprising 8 columns extracted from its "data" and "close" columns. The input consists of a vector representing the closing prices of the previous 7 days, while the target represents the closing price of the current day. The data is divided into 16 batches.

4.3 MODEL'S ARCHITECTURE

The model consists of two LSTM layers, each comprising 4 nodes, followed by a dense layer for the output.

[4] https://drive.google.com/file/d/1MqY9yaql1XQbodFSngsHxGbyLdWRhVXj/view

4.4 LOSS FUNCTIONS:

Mean Squared Error also known as the L_2 loss, is a common loss function used in regression tasks. It quantifies the magnitude of the error between a model's predictions $\hat{y}_i$ and the actual output y_i. It's given by:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

4.5 EVALUATION METRICS

Mean Absolute Error (MAE): measures the average magnitude of the errors in a set of predictions, without considering their direction. It's computed using:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where:

n is the number of observations.

y_i are the actual values and $\hat{y}_i$ are the predicted values.

Mean Squared Error (MSE): measures the average squared difference between the predicted values $\hat{y}_i$ and the actual values y_i. Its formula is :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Coefficient of determination: or R^2, indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, with 1 signifying that the regression predictions perfectly fit the data. It can be calculated using:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Such that: $\bar{y}$ represents the mean of the observed values y_i and $\hat{y}_i$ represents the predicted values.

Mean Absolute Percentage Error (MAPE): measures the average absolute percentage difference between actual y_i and predicted $\hat{y}_i$ values.Its formula is:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\%$$

Table 3.1: Summaries of the models utilized for our experiments.

Task	Model	Dataset
Image Classification	CNN ResNet34	MNIST CIFAR 10
Language modeling	3 LSTM Layers	Penn Treebank
Time Series Forecasting	2 LSTM Layers	Amazon Stock

5 TRAINING

We utilize Kaggle platforms for conducting our experiments with the following hardware specifications:
GPU : NVIDIA TESLA P100 GPU and NVIDIA TESLA T4 x2 GPU
RAM: 15GB memory.

Every model was trained with each of the 15 optimizers, using the recommended default hyperparameters: $\alpha = 10^{-3}$, $(\beta_1, \beta_2, \beta_3) = (0.9, 0.999, 0.9999)$, $\epsilon = 10^{-8}$, for AdamW we use $\lambda = 10^{-2}$ as weight decay, also RoAdam's thresholds are $(k, K) = (0.1, 10)$, and for T-Adam we use $\nu = 1$,$d = dim(\nabla_\theta f_t(\theta))$. For AdaBound we use as boundaries this two functions:

$$\eta_l(t) = \left(1 - \frac{1}{(1-\gamma)(t+1)}\right)\alpha$$

$$\eta_u(t) = \left(1 + \frac{1}{(1-\gamma)t}\right)\alpha$$

With $\gamma = 10^{-3}$

6 CONCLUSION

In this chapter, we outline the experimental settings for three different tasks employed in this study. We begin with a description of the datasets utilized in each task, along with their preprocessing steps. Subsequently, we detail the model architectures, the choice of loss functions, and the evaluation metrics employed. The next chapter will explore and analyze the obtained results.

CHAPTER 4

EXPERIMENTAL RESULTS

1 INTRODUCTION

In this chapter, we'll offer a comparative study of optimization methods rooted in Adam to underscore how different optimizers impact the efficacy and performance of deep learning models.

2 RESULTS AND DISCUSSION

We present the results of our experiments and provide a detailed discussion within the context of specific tasks, for each category, and in relation to other categories.

2.1 IMAGE CLASSIFICATION TASK

The tables (4.1,4.2) present the accuracy, recall, precision, F1-score, MCC and training time of training CNN and ResNet34 models with 14 optimizers respectively. The figures (4.1,4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10,4.11, 4.12, 4.13, 4.14) and (4.15, 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, 4.22, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28) provides a visual representation of the training loss and accuracy achieved by each optimizer.

Table 4.1: The results of training CNN using MNIST dataset for 150 epochs

Optimizers	Accuracy	Recall	Precision	F_1	MCC	Training Time
Adam	98.59	0.9859	0.9859	0.9858	0.9843	**24 min 50 sec**
AdaMod	**98.80**	**0.988**	**0.9880**	**0.9879**	**0.9866**	29 min 1 sec
AdaBound	98.52	0.9852	0.9852	0.9851	0.9835	27 min 56 sec
R-Adam	98.76	0.9876	0.9876	0.9875	0.9862	25 min 9 sec
AdaMax	98.42	0.9842	0.9842	0.9841	0.9824	27 min 14 sec
AMSGrdad	98.52	0.9852	0.9852	0.9851	0.9835	26 min 31 sec
E-Adam	98.44	0.9844	0.9844	0.9844	0.9826	26 min 52 sec
AdaBelief	98.49	0.9849	0.9849	0.9848	0.9832	27 min 35 sec
YOGI	98.51	0.9851	0.9851	0.9850	0.9834	28 min 37 sec
AdamW	98.76	0.9876	0.9876	0.9875	0.9862	24 min 54 sec
N-Adam	98.45	0.9845	0.9846	0.9844	0.9827	25 min 22 sec
ND-Adam	98.64	0.9864	0.9864	0.9863	0.9848	26 min 23 sec
MSVAG	98.21	0.9821	0.9821	0.9820	0.9801	32 min 26 sec
T-Adam	98.44	0.9844	0.9844	0.9843	0.9826	32 min 49 sec

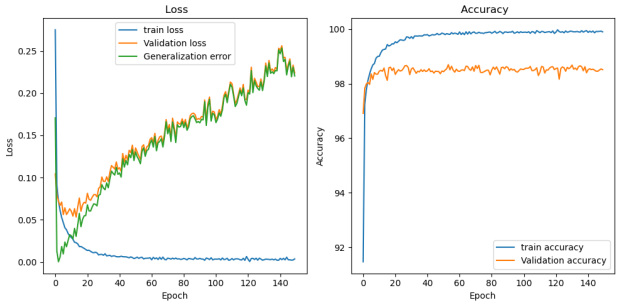

Figure 4.1: Adam with CNN

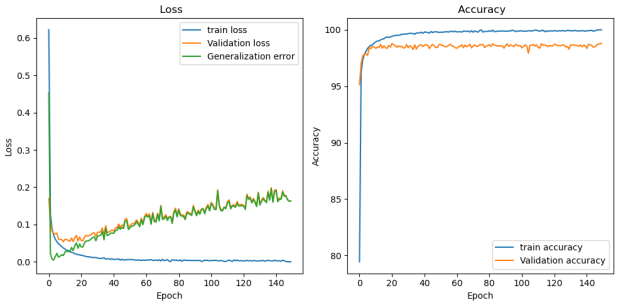

Figure 4.2: AdaMod with CNN

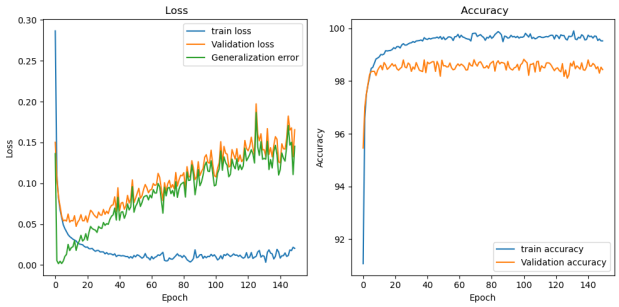

Figure 4.3: AdaBound with CNN

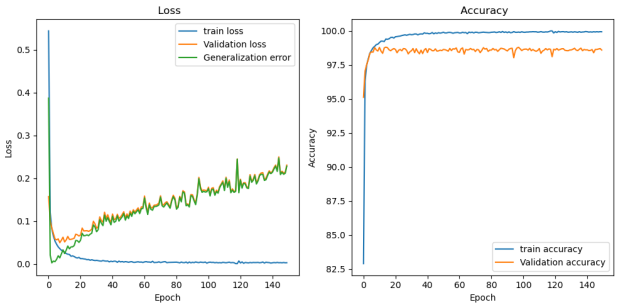

Figure 4.4: R-Adam with CNN

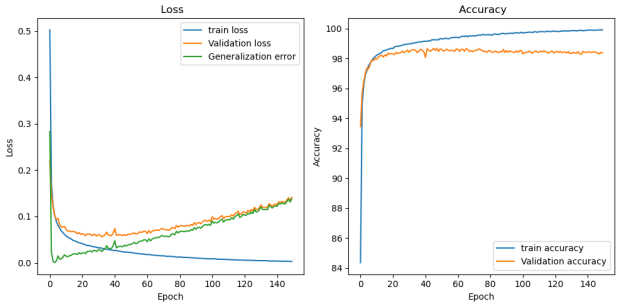

Figure 4.5: AdaMax with CNN

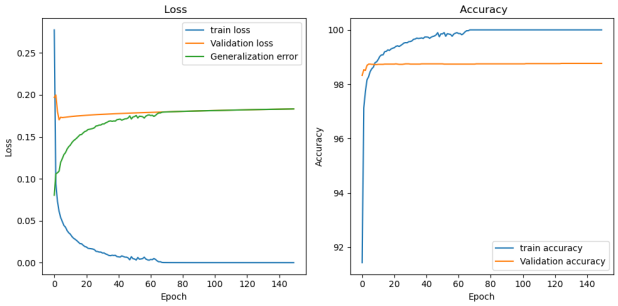

Figure 4.6: AMSGrad with CNN

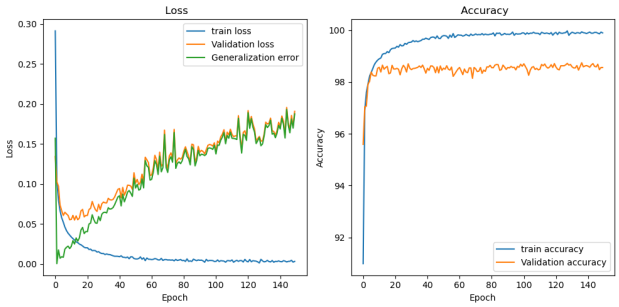

Figure 4.7: E-Adam with CNN

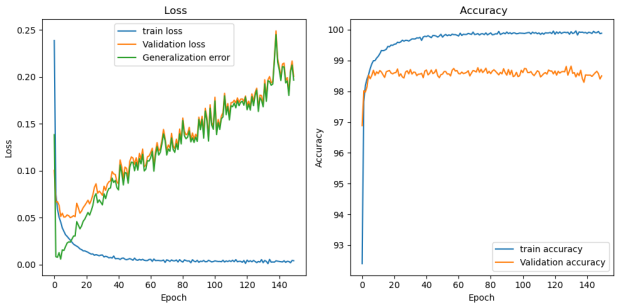

Figure 4.8: AdaBelief with CNN

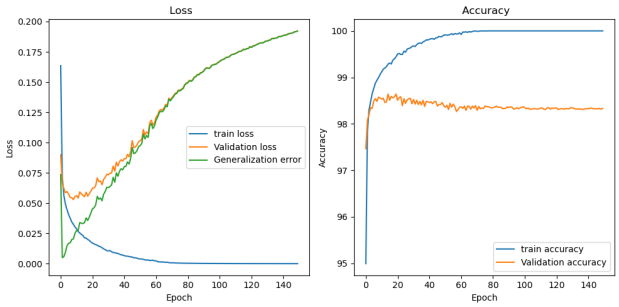

Figure 4.9: YOGI with CNN

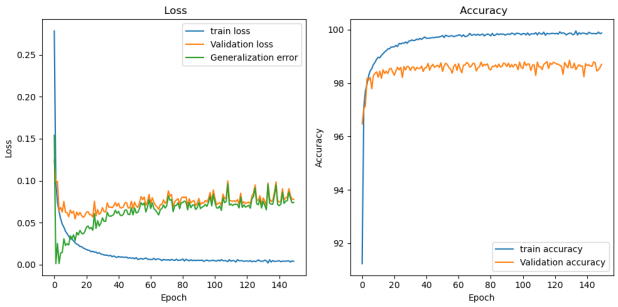

Figure 4.10: AdamW with CNN

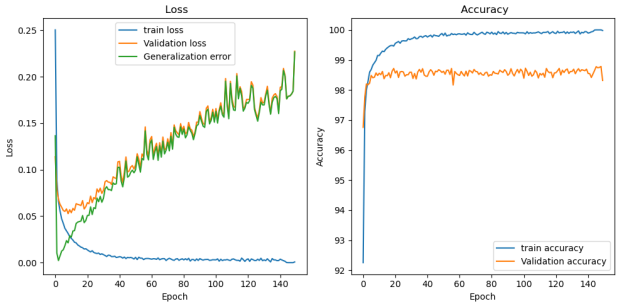

Figure 4.11: N-Adam with CNN

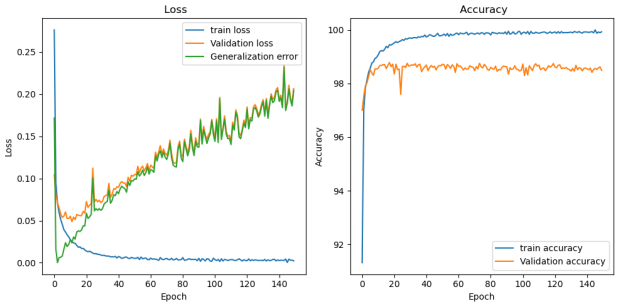

Figure 4.12: ND-Adam with CNN

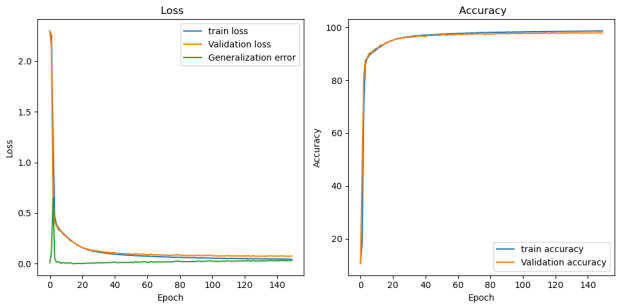

Figure 4.13: MSVAG with CNN

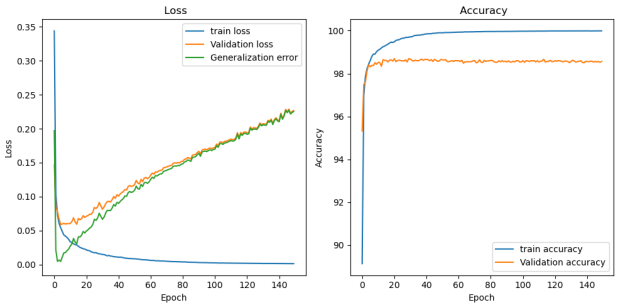

Figure 4.14: TAdam with CNN

2.1.1 CNN MODEL

We can observe from training a simple model that:

- From the table 4.1 and the figure4.2, it is evident that AdaMod outperforms the other optimizers, irrespective of the time required. Additionally, it effectively addresses the overfitting problem, as demonstrated by the favorable loss function curve.

- The Learning Rate Modifiers outperforms the others with closely comparable results, likely because of their shared strategy of adjusting the learning rate during the training process, which results in reduced fluctuations in the loss function curve as illustrated in figures: 4.2,4.3,4.4 .

- The performance of the Enhanced Velocity Control optimizers has been notably poorer compared to other optimizers, particularly Adam. Additionally,These methods also demonstrate susceptibility to overfitting, as evidenced by their significant generalization gap. This can be attributed to the inadequacy of their modifications to the second moment of Adam, particularly when applied to simpler models such as our CNN.

- In the Adam Corporation Enhanced category, AdamW and NDAdam outperform the standard Adam optimizer with only minor differences. This is may be attributed to the enhancements introduced by AdamW, which incorporates a regularization term into the updating rule of Adam. Similarly, NDAdam optimizes the updating direction, contributing to its improved performance compared to traditional Adam.

- For the Noise Sensitivity Reducers, The replacement of the Gaussian distribution in T-Adam doesn't significantly impact performance increases the processing time, which is why T-Adam's performance is similar to Adam for this model. However, it helps in reducing fluctuations in the loss curve figure 4.14.

- Adam proves to be the quickest optimizer among its variants, while T-Adam lags behind, taking approximately 8 minutes longer.

Table 4.2: The results of training ResNet using CIFAR10 dataset for 150 epochs

Optimizers	Accuracy	Recall	Precision	F_1	MCC	Training Time
Adam	71.48	0.7144	0.7165	0.7141	0.6829	**90 min 10 sec**
AdaMod	70.53	0.7052	0.7092	0.7092	0.6727	126 min 17 sec
AdaBound	72.44	0.7243	0.7235	0.7236	0.6937	111 min 46 sec
R-Adam	70.08	0.7006	0.7031	0.7008	0.6675	94 min 20 sec
AdaMax	69.23	0.6923	0.6952	0.6923	0.6584	98 min 29 sec
AMSGrdad	**73.41**	**0.734**	**0.7338**	**0.7336**	**0.7045**	90 min 54 sec
E-Adam	71.37	0.7136	0.7178	0.7139	0.6822	100 min 57 sec
AdaBelief	71.69	0.7169	0.7192	0.7170	0.6856	114 min 39 sec
YOGI	69.51	0.6951	0.6945	0.6944	0.6613	109 min 41 sec
AdamW	71.59	0.716	0.7213	0.7149	0.6852	94 min 4 sec
N-Adam	62.81	0.628	0.6301	0.6283	0.5868	96 min 17 sec
ND-Adam	69.57	0.6956	0.7078	0.6956	0.6630	94 min 32 sec
MSVAG	60.03	0.6005	0.5986	0.5993	0.5561	147 min 54 sec
T-Adam	72.38	0.7239	0.7223	0.7228	0.6932	201 min 27 sec

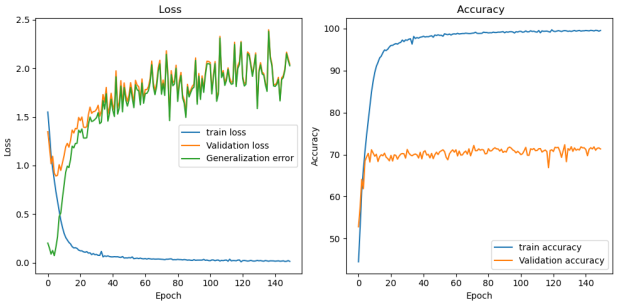

Figure 4.15: Adam with ResNet

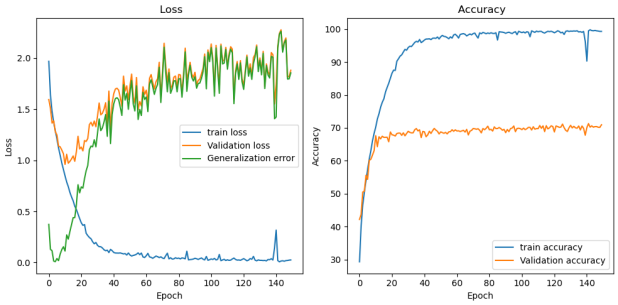

Figure 4.16: AdaMod with ResNet

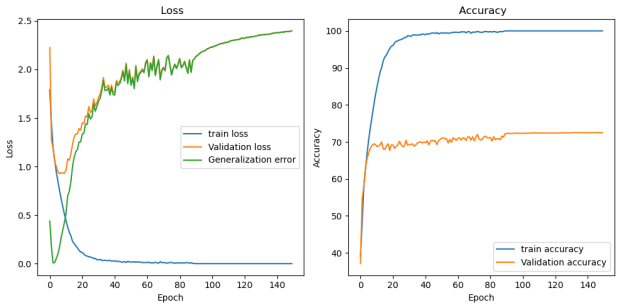

Figure 4.17: AdaBound with ResNet

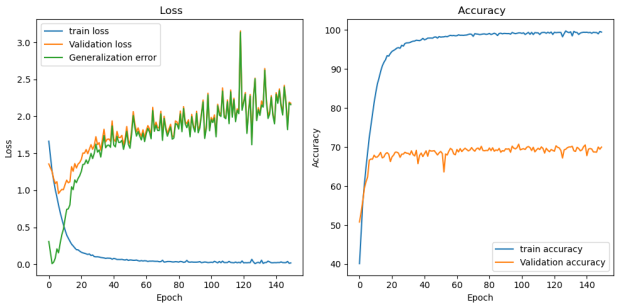

Figure 4.18: R-Adam with ResNet

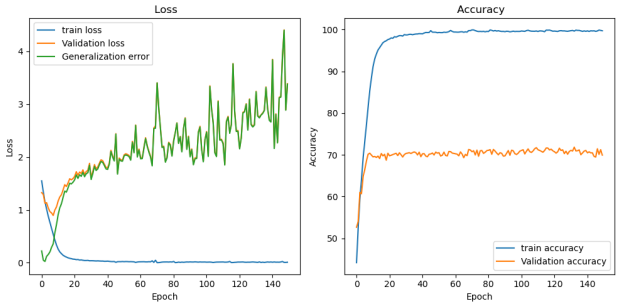

Figure 4.19: AdaMax with ResNet

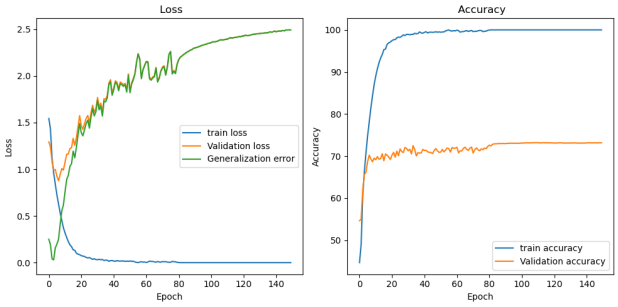

Figure 4.20: AMSGrad with ResNet

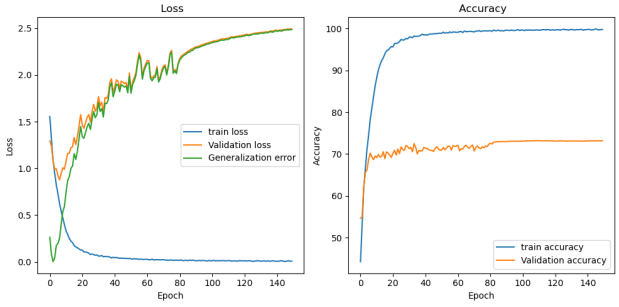

Figure 4.21: E-Adam with ResNet

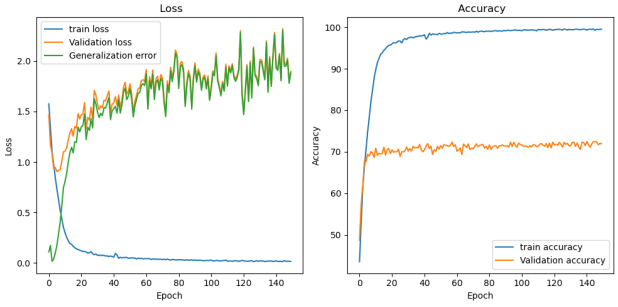

Figure 4.22: AdaBelief with ResNet

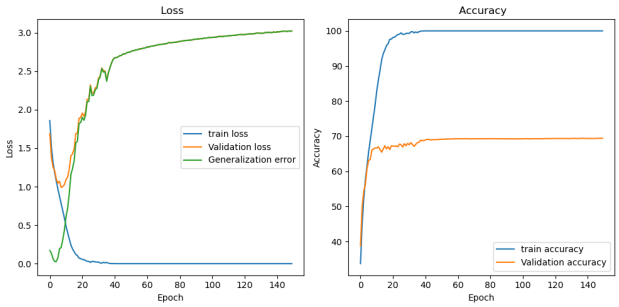

Figure 4.23: YOGI with ResNet

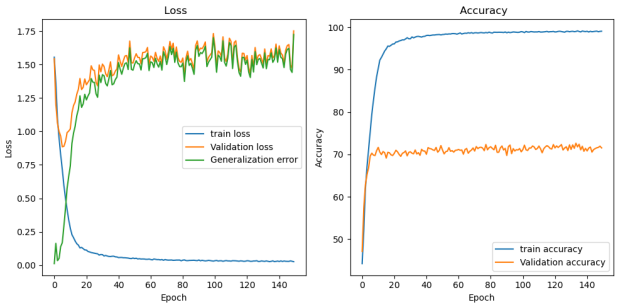

Figure 4.24: AdamW with ResNet

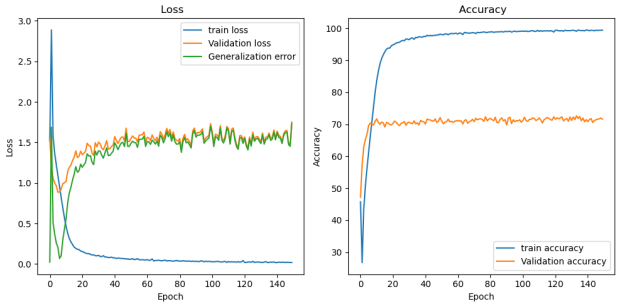

Figure 4.25: N-Adam with ResNet

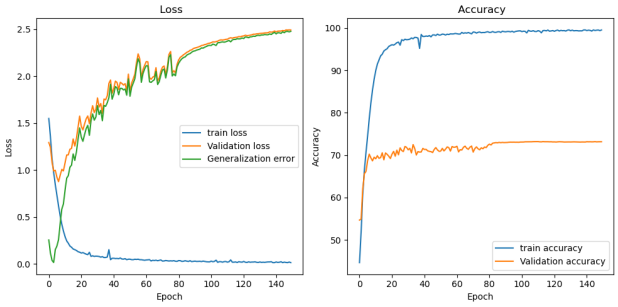

Figure 4.26: ND-Adam with ResNet

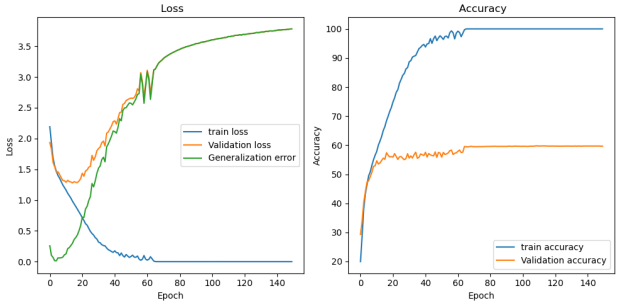

Figure 4.27: MSVAG with ResNet

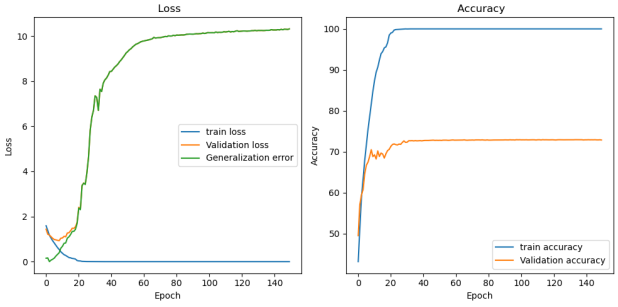

Figure 4.28: T-Adam with ResNet

2.1.2 RESNET34 MODEL

For a deep model like ResNet, we found different results and observed that:

- Overall, the AMSGrad optimizer demonstrates superior performance compared to other optimizers, ranking as the second fastest after Adam. This superiority is likely due to its ability to control the step size by modifying the second moment in Adam. Following AMSGrad, AdaBound and T-Adam perform similarly, with only a slight difference in their effectiveness.

- All optimizers face challenges with overfitting and are computationally intensive; they perform well on the training data but fail to generalize to the validation data. This issue is particularly pronounced in T-Adam, which exhibits a substantial gap in generalization and requires extended processing time.

- For the Learning Rate Modifiers, AdaBound demonstrates outstanding performance, ranking as the second fastest optimizer after R-Adam. Its technique for controlling the learning rate makes it more stable during the training process.

- In the Enhanced Velocity Control category, two optimizers, AMSGrad (the fastset) and AdaBelief (the slowest) demonstrate superior performance compared to Adam. They address overfitting by controlling the learning rate through modification of the second moment using historical gradient information.

- Similar to the previous model (CNN), AdamW demonstrates significantly improved performance and speed compared to the optimizers in the Adam Corporation Enhanced category.However, T-Adam shows better performance than Adam when applied to ResNet, regardless of the duration, suggesting that its technique may be particularly effective in deep models.

- There is a significant time difference between the fastest and slowest optimizer, approximately 111 minutes longer.

2.2 LANGUAGE MODELING TASK

The table 4.3 represent the accuracy, perplexity, F_1 scores and training time obtained by training LSTM model on Penn Treebank dataset for 200 epoch. the highest value of the

accuracy and f_1 score the lowest value of the perplexity indicate the well trained optimizer. the figures (4.29,4.30,4.31,4.32,4.33,4.34,4.35, 4.36,4.38,4.39,4.40) visualize the perplexity and accuracy curves.

Table 4.3: The results of training LSTM using Penn Treebank dataset for 200 epochs

Optimizers	Accuracy	Perplexity	F_1	Training Time
Adam	92.52	1.872	0.9194	77 min 39 sec
AdaMod	92.29	1.949	0.9171	89 min 30 sec
AdaBound	92.06	1.781	0.9137	83 min 38 sec
R-Adam	92.53	1.881	0.9197	79 min 40 sec
AdaMax	92.41	1.865	0.9173	81 min 43 sec
AMSGrdad	92.06	3.127	0.9207	78 min 3 sec
E-Adam	92.28	1.768	0.9158	80 min 38 sec
AdaBelief	92.56	2.127	0.9196	83 min 38 sec
YOGI	92.17	1.869	0.9158	84 min 6 sec
AdamW	**92.71**	**1.760**	**0.9220**	**77 min 28 sec**
N-Adam	92.38	1.918	0.9180	79 min 37 sec
ND-Adam	92.38	1.929	0.9180	78 min 25 sec
MSVAG	91.06	2.031	0.9	96 min 53 sec
T-Adam	92.26	1.906	0.916	118 min 13 sec

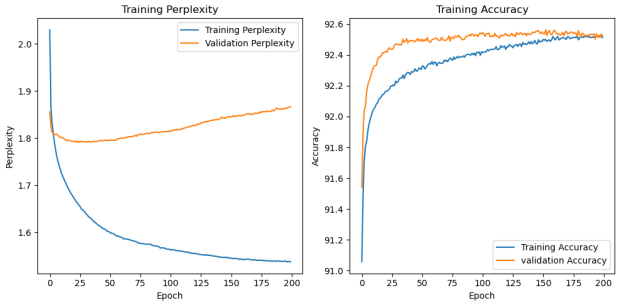

Figure 4.29: Adam with LSTM

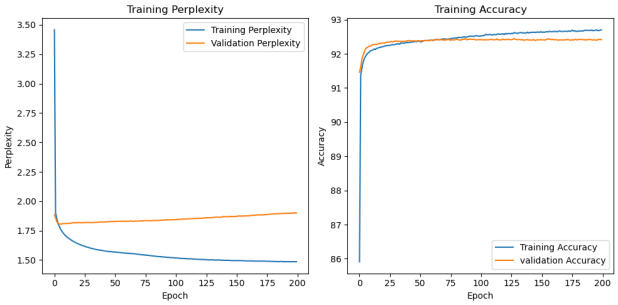

Figure 4.30: AdaMod with LSTM

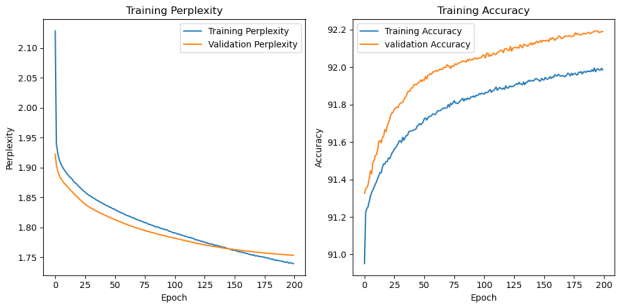

Figure 4.31: AdaBound with LSTM

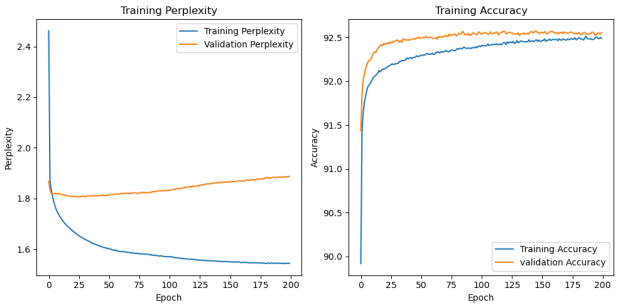

Figure 4.32: R-Adam with LSTM

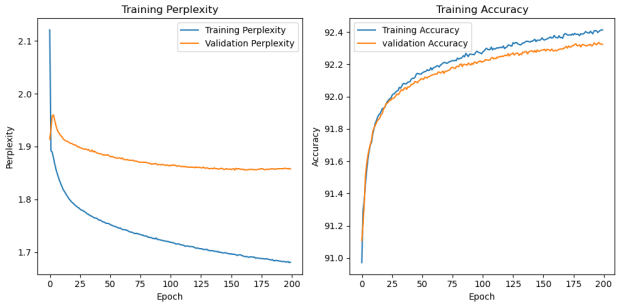

Figure 4.33: AdaMax with LSTN

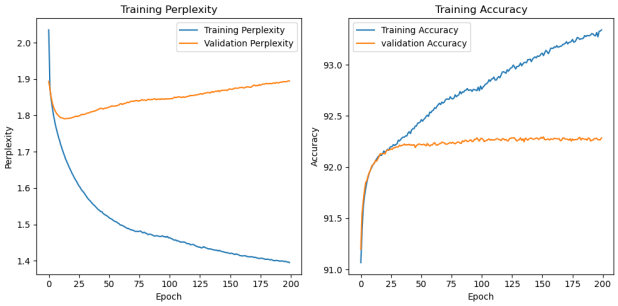

Figure 4.34: AMSGrad with LSTM

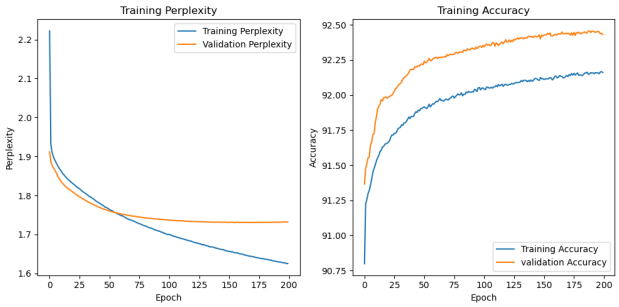

Figure 4.35: E-Adam with LSTM

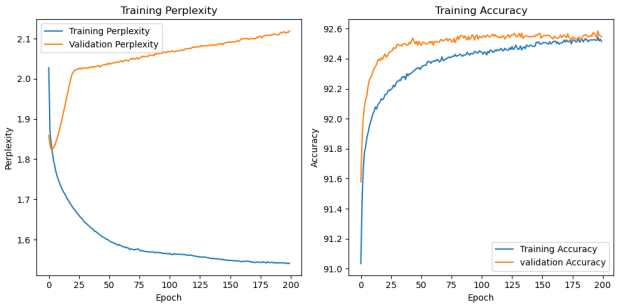

Figure 4.36: AdaBelief with LSTM

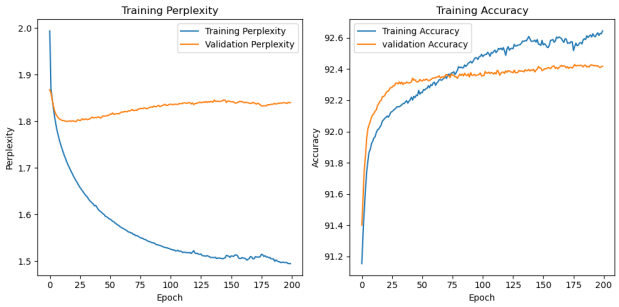

Figure 4.37: YOGI with LSTM

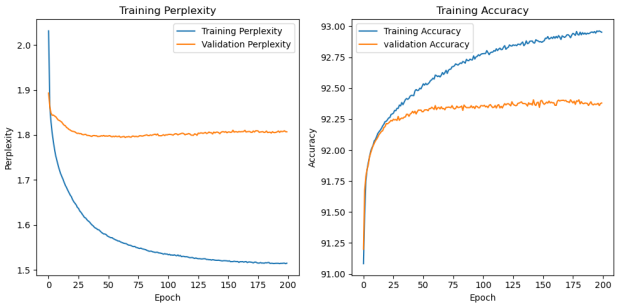

Figure 4.38: AdamW with LSTM

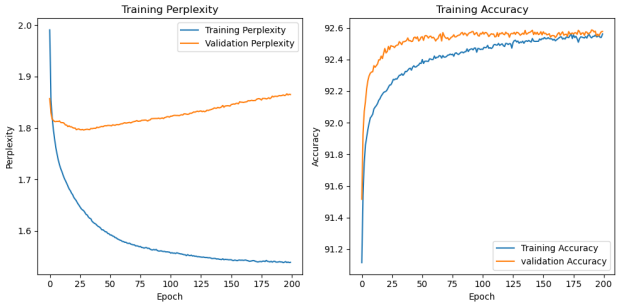

Figure 4.39: N-Adam with LSTM

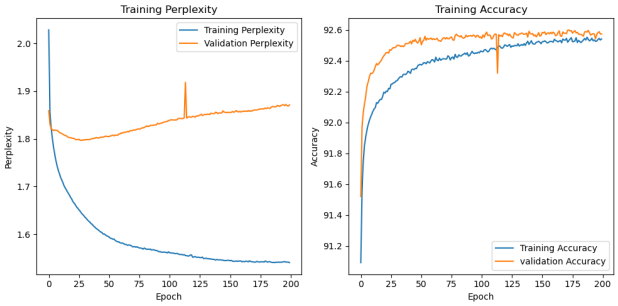

Figure 4.40: ND-Adam with LSTM

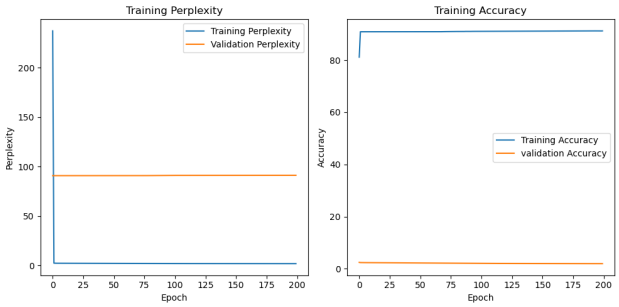

Figure 4.41: MSVAG with LSTM

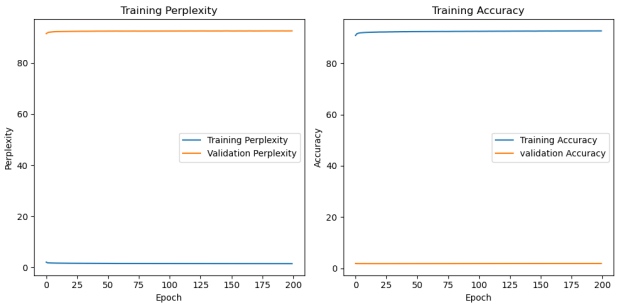

Figure 4.42: T-Adam with LSTM

2.2.1 DISCUSSION

- The AdamW optimizer achieves better performance and operates more efficiently by directly applying weight decay to the parameters θ, which results in more accurate

gradient updates and reduced overfitting. Following AdamW, RAdam and AdaBelief also outperform the traditional Adam optimizer, offering better stability and improved training dynamics.

- All of these optimizers have closely comparable scores, but most perform worse than Adam. This suggests that their modifications may not be effective for this specific task or data.

- For the Learning Rate Modifiers, R-Adam stands out as the top performer. This superior performance is likely due to its rectifier term which mitigates the high variance of learning rates, especially during the initial training phases, which helps in achieving stable convergence. However, the model trained with Adabound exhibits a lower perplexity score, indicating proficiency in predicting and generating text solely within the confines of the training data.

- Adabelief optimizes step sizes effectively through its incorporation of a belief term in the second moment, resulting in the highest accuracy and F_1 scores among the optimizers in the Enhanced Velocity Control category. Conversely, E-Adam achieves the top perplexity score, indicating its strong generalization capabilities generating text.

- The Adam Corporation Enhanced optimizers are among the fastest compared to others, closely approaching the speed and performance of Adam. However, MSVAG is time-consuming and performs poorly.

2.3 TIME SERIES FORECASTING TASK

MAE, MSE, R2, and MAPE metrics and the training time are collected in the table4.4 after training the LSTM model on the Amazon stock dataset with 15 optimizers. The lowest values for MAE, MSE, and MAPE indicate that the model is well trained, while the highest value for R2 also signifies good model performance. The following figures illustrate the training loss curve(right) and predictions(left) for each optimizer.

Table 4.4: The results of training LSTM using Amazon Stock dataset for 150 epochs

Optimizers	MAE	MSE	R^2	MAPE	Training Time
Adam	0.0293	0.0016	0.9780	34.74	**1 min 48 sec**
AdaMod	0.0313	0.0017	0.9764	46.49	2 min 52 sec
AdaBound	0.0411	0.0028	0.9624	53.46	2 min 31 sec
R-Adam	0.0288	**0.0015**	0.9787	37.28	1 min 51 sec
AdaMax	0.039	0.0025	0.9656	51.65	2 min 18 sec
AMSGrdad	0.0417	0.0029	0.9601	54.19	1 min 49 sec
E-Adam	0.0294	0.0016	0.9786	34.45	2 min 19 sec
AdaBelief	**0.0286**	**0.0015**	0.9787	35.34	2 min 32 sec
YOGI	0.0352	0.0020	0.9724	39.78	2 min 23 sec
AdamW	0.0294	0.0016	0.9786	**34.45**	1 min 49 sec
N-Adam	0.0293	0.0016	0.9781	37.82	1 min 53 sec
ND-Adam	0.0289	**0.0015**	0.9787	35.36	2 min 10 sec
MSVAG	0.1258	0.0211	0.7177	331.27	3 min 16 sec
T-Adam	0.0287	**0.0015**	**0.9792**	35.47	3 min 52 sec
Ro-Adam	0.0353	0.0021	0.9718	41.84	3 min 39 sec

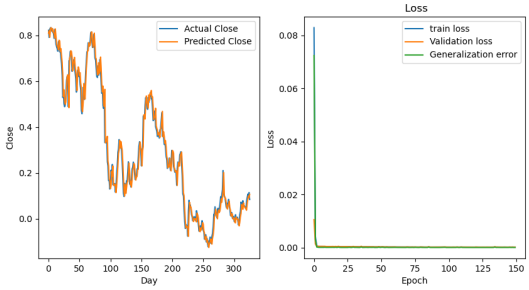

Figure 4.43: Adam with LSTM

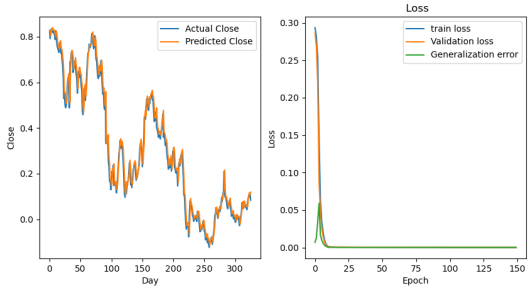

Figure 4.44: AdaMod with LSTM

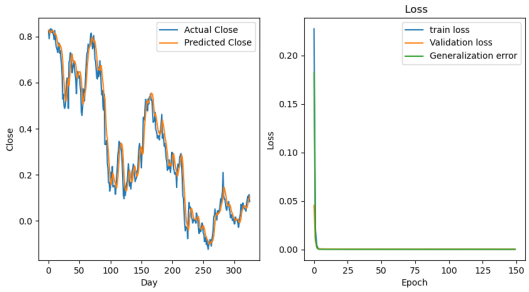

Figure 4.45: AdaBound with LSTM

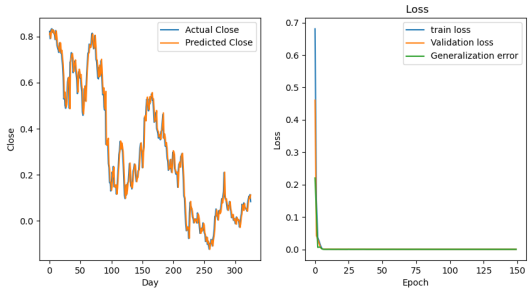

Figure 4.46: R-Adam with LSTM

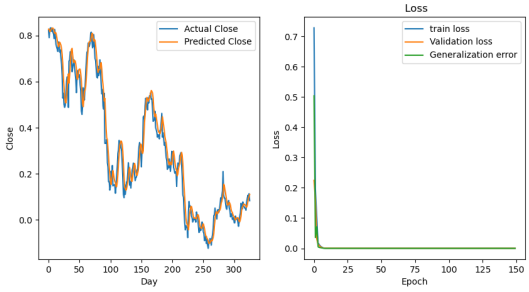

Figure 4.47: AdaMax with LSTN

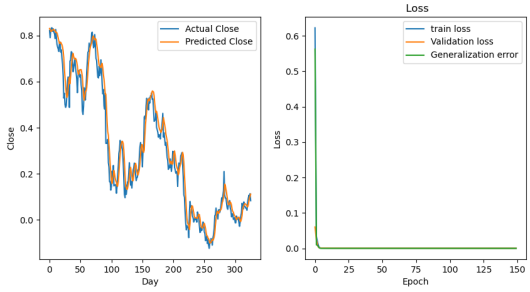

Figure 4.48: AMSGrad with LSTM

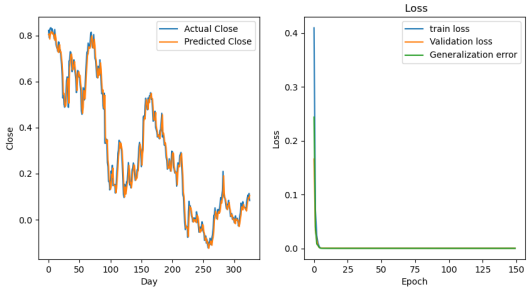

Figure 4.49: E-Adam with LSTM

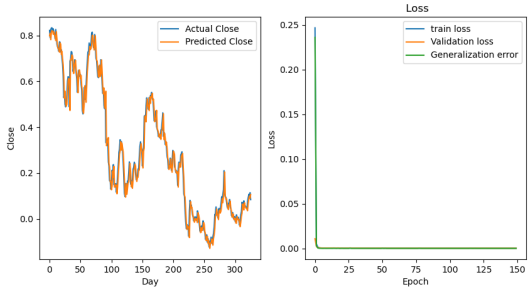

Figure 4.50: AdaBelief with LSTM

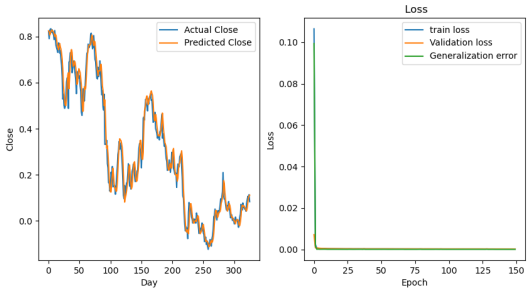

Figure 4.51: YOGI with LSTM

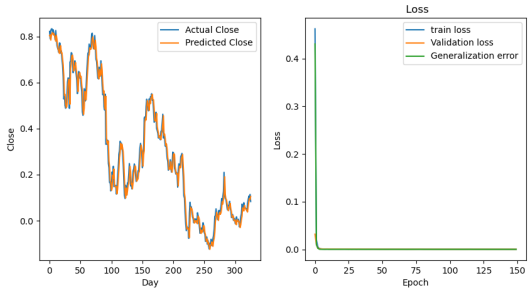

Figure 4.52: AdamW with LSTM

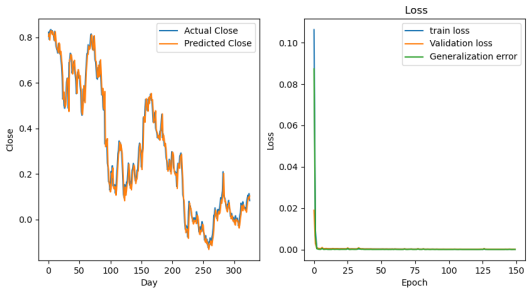

Figure 4.53: N-Adam with LSTM

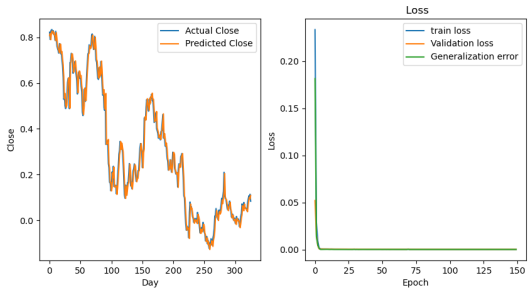

Figure 4.54: ND-Adam with LSTM

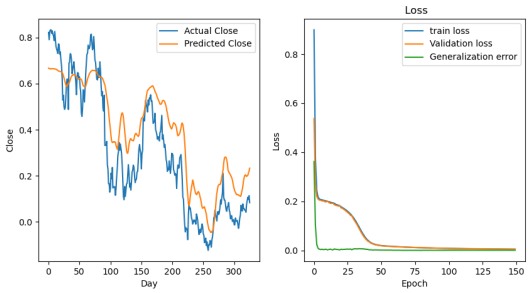

Figure 4.55: MSVAG with LSTM

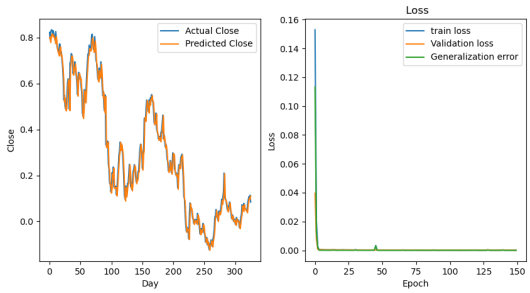

Figure 4.56: TAdam with LSTM

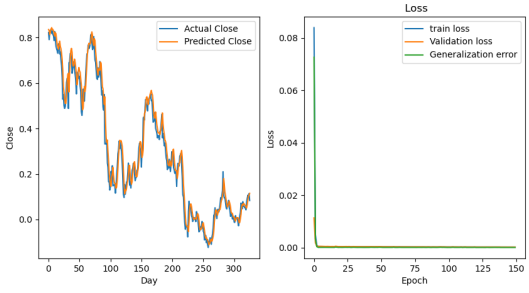

Figure 4.57: RoAdam with LSTM

2.3.1 DISCUSSION

- For the MAE score, AdaBelief has the lowest value, indicating it performs well regardless of the direction of the errors. Additionally, T-Adam achieves the best score in the R2 metric, indicating it fits the data well with very small variance. However, for the MARE score, AdamW has the lowest value, demonstrating its reliability in predictions.

- Seven out of fourteen optimizers exhibit superior performance compared to Adam, including R-Adam, AdaBelief, E-Adam, AdamW, N-Adam, ND-Adam, and T-Adam. These optimizers demonstrate excellent data fitting capabilities, yielding predictions that closely align with the actual data.

- The MSVAG optimizer displays the poorest performance, struggling to accurately fit the data and yielding random predictions. Conversely, the other optimizers in the Adam Corporation Enhanced category demonstrate strong performance, consistently outperforming expectations.

- for the Noise Sensitivity Reducers, T-Adam achieves best performance due to its robust properties.

Here, we summarize the key findings of this study:

- The MSVAG optimizer demonstrates the poorest performance and is time-consuming among the three tasks, facing challenges in accurately fitting the data and producing unpredictable predictions. This could be attributed to the sensitivity of its performance to hyper-parameter tuning.

- AdaMod and AMSGrad excel in optimizing neural networks for image classification tasks, showcasing superior performance. However, Adam remains the fastest optimizer.

- AdamW stands out as the premier option for language modeling tasks. Its weight decay mechanism significantly improves the model's capacity to generalize and produce coherent text, outperforming other Adam variants in both perplexity scores and predictive accuracy.

- AdaBelief and T-Adam demonstrate exceptional performance in time series forecasting despite the significant amount of time required. These variants excel in capturing temporal dependencies and reliably predicting future trends, making them particularly effective for models tailored to time series data.

3 CONCLUSION

In conclusion, this comparative study has underscored the diverse strengths of various Adam variants across different tasks and emphasized the criticality of choosing an optimizer that aligns with the specific characteristics and demands of each task.

General Conclusion

This thesis aims to evaluate the performance of Adam variants across a diverse set of datasets and tasks to better understand the generalizability of each optimizer. By conducting a comparative study on three distinct tasks—image classification, language modeling, and time series forecasting—we identify the optimal Adam variant and elucidate the strengths and weaknesses of each optimizer.

In the first chapter, we offered an overview of various architectures applied to different tasks pertinent to this study. We emphasized the significance of optimization methods in training deep learning models.

In the second chapter, we provide a detailed overview of the Adam algorithm, discuss its limitations, and introduce several of its variants, categorized by their modifications and the specific issues they address.

The third chapter details a comparative study on image classification, language modeling, and time series forecasting, describing datasets, preprocessing methods, training configurations, and performance metrics.

The last chapter presents a comparative analysis of Adam-based optimizers to demonstrate how these optimizers impact the effectiveness and performance of deep learning models.

We could not include certain Adam variations in this comparative analysis due to time and technical limitations. Our future work will explore these variations more extensively with additional optimizers.

BIBLIOGRAPHY

1. Balles, L. & Hennig, P. Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients. *arXiv preprint arXiv:1705.07774* (2017).
2. Bengio, Y., Goodfellow, I. & Courville, A. *Deep learning* (MIT press Cambridge, MA, USA, 2017).
3. Braiek, H. B. & Khomh, F. Machine Learning Robustness: A Primer. *arXiv preprint arXiv:2404.00897* (2024).
4. Choi, D. *et al.* On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446* (2019).
5. Darken, C., Chang, J., Moody, J., *et al. Learning rate schedules for faster stochastic gradient search* in *Neural networks for signal processing* **2** (1992), 3–12.
6. Diederik, P. K. Adam: A method for stochastic optimization. *(No Title)* (2014).
7. Ding, J, Ren, X, Luo, R & Sun, X. An adaptive and momental bound method for stochastic learning. arXiv 2019. *arXiv preprint arXiv:1910.12249*.
8. Dogo, E. M., Afolabi, O., Nwulu, N., Twala, B. & Aigbavboa, C. *A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks* in *2018 international conference on computational techniques, electronics and mechanical systems (CTEMS)* (2018), 92–99.
9. Dozat, T. Incorporating nesterov momentum into adam (2016).

10. Duchi, J., Hazan, E. & Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* **12** (2011).

11. EITCA Academy. *Why do we need to apply optimizations in machine learning?* Accessed: 2024-06-14. 2024. https://eitca.org/artificial-intelligence/eitc-ai-adl-advanced-deep-learning/optimization/optimization-for-machine-learning/why-do-we-need-to-apply-optimizations-in-machine-learning/.

12. Fatima, N. *et al.* Enhancing performance of a deep neural network: A comparative analysis of optimization algorithms (2020).

13. Glorot, X., Bordes, A. & Bengio, Y. *Deep sparse rectifier neural networks* in *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), 315–323.

14. Hardt, M., Recht, B. & Singer, Y. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240* (2016).

15. He, K., Zhang, X., Ren, S. & Sun, J. *Deep residual learning for image recognition* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 770–778.

16. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9,** 1735–1780 (1997).

17. Ilboudo, W. E. L., Kobayashi, T. & Sugimoto, K. Tadam: A robust stochastic gradient optimizer. *arXiv preprint arXiv:2003.00179* (2020).

18. Jiang, Y. & Zhou, Q. Understanding the Generalization of Adam in Learning Neural Networks with Proper Regularization. *arXiv preprint arXiv:2105.05717* (2021).

19. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86,** 2278–2324 (1998).

20. Liu, L. *et al.* On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265* (2019).

21. Loshchilov, I. & Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

22. Loshchilov, I. & Hutter, F. *Decoupled weight decay regularization* in *Proceedings of the Sixth International Conference on Learning Representations* (2018).

23. Loshchilov, I. & Hutter, F. Fixing weight decay regularization in adam (2018).

24. Luo, L., Xiong, Y., Liu, Y. & Sun, X. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843* (2019).

25. Meister, C. & Cotterell, R. Language model evaluation beyond perplexity. *arXiv preprint arXiv:2106.00085* (2021).

26. Montesinos López, O. A., Montesinos López, A. & Crossa, J. in *Multivariate statistical machine learning methods for genomic prediction* 109–139 (Springer, 2022).

27. Nesterov, Y. *A method for unconstrained convex minimization problem with the rate of convergence O (1/k2)* in *Dokl. Akad. Nauk. SSSR* **269** (1983), 543.

28. Pascanu, R., Mikolov, T. & Bengio, Y. *On the difficulty of training recurrent neural networks* in *International conference on machine learning* (2013), 1310–1318.

29. Qian, N. On the momentum term in gradient descent learning algorithms. *Neural networks* **12,** 145–151 (1999).

30. Reddi, S. J., Kale, S. & Kumar, S. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237* (2019).

31. Robbins, H. & Monro, S. A stochastic approximation method. *The annals of mathematical statistics,* 400–407 (1951).

32. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

33. Shazeer, N. & Stern, M. *Adafactor: Adaptive learning rates with sublinear memory cost* in *International Conference on Machine Learning* (2018), 4596–4604.

34. Solanke, A. V. & Patnaik, G. K. Intrusion detection using deep learning approach with different optimization. *International Journal for Research in Applied Science and Engineering Technology* **8,** 128–134 (2020).

35. Soydaner, D. A comparison of optimization algorithms for deep learning. *International Journal of Pattern Recognition and Artificial Intelligence* **34,** 2052013 (2020).

36. Sun, S., Cao, Z., Zhu, H. & Zhao, J. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics* **50,** 3668–3681 (2019).

37. Tao, H. & Lu, X. *On comparing six optimization algorithms for network-based wind speed forecasting* in *2018 37th Chinese Control Conference (CCC)* (2018), 8843–8850.

38. Tato, A. & Nkambou, R. Improving adam optimizer (2018).

39. Tieleman, T. & Hinton, G. Rmsprop: Divide the gradient by a running average of its recent magnitude. *coursera: Neural networks for machine learning* **17** (2012).

40. *Unlocking the Power of Iterative Algorithms: A Deep Dive* https://locall.host/is-algorithm-iterative/. Accessed: 2024-06-14. 2024.

41. Voulodimos, A., Doulamis, N., Doulamis, A. & Protopapadakis, E. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience* (2018).

42. Wang, X. *et al.* Edge AI: Convergence of Edge Computing and Artificial Intelligence. *Journal or Conference Name* **Volume Number,** Page Range (2020).

43. Wang, Y. *et al.* Rethinking Adam: A twofold exponential moving average approach. *arXiv preprint arXiv:2106.11514* (2021).

44. Wilson, A. C., Roelofs, R., Stern, M., Srebro, N. & Recht, B. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems* **30** (2017).

45. Yang, H., Pan, Z. & Tao, Q. Robust and adaptive online time series prediction with long short-term memory. *Computational intelligence and neuroscience* **2017** (2017).

46. Yuan, W. & Gao, K.-X. EAdam Optimizer: How ϵ Impact Adam. *arXiv preprint arXiv:2011.02150* (2020).

47. Zaheer, M., Reddi, S., Sachan, D., Kale, S. & Kumar, S. Adaptive methods for nonconvex optimization. *Advances in neural information processing systems* **31** (2018).

48. Zargar, S. Introduction to sequence learning models: RNN, LSTM, GRU. *Department of Mechanical and Aerospace Engineering, North Carolina State University* (2021).

49. Zhang, Z. *Improved adam optimizer for deep neural networks* in *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)* (2018), 1–2.

50. Zhuang, J. *et al.* Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems* **33,** 18795–18806 (2020).

51. Zou, D., Cao, Y., Li, Y. & Gu, Q. Understanding the generalization of adam in learning neural networks with proper regularization. *arXiv preprint arXiv:2108.11371* (2021).