Extending the Work of DT-Fixup: Examining the Effects of PowerNorm and MADGRAD Optimization on DT-Fixup Performance

Prem Shankar Mohan
University of Windsor

Extending the work of DT-Fixup: Examining the Effects of PowerNorm and MADGRAD Optimization on DT-Fixup Performance

By

Prem Shanker Mohan

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2023

Extending the work of DT-Fixup: Examining the Effects of PowerNorm and MADGRAD Optimization on DT-Fixup Performance

by

Prem Shanker Mohan

APPROVED BY:

June 05, 2023

ABSTRACT

With the introduction of the attention technique, the Bidirectional Encoder Representations from Transformers (BERT) have greatly advanced the study of solving sequence-to-sequence tasks in Natural Language Processing (NLP). When the task-specific annotations are limited, the NLP tasks are commonly performed by pre-training a model using the transformer technique on large-scale general corpora, followed by fine-tuning the model on domain-specific data. Instead of using shallow neural components for fine-tuning, additional transformer layers could be introduced into the architecture. Recent research shows that, by resolving some initialization and optimization issues, these augmented transformer layers could lead to performance gains despite of the limited size of the available data, and this can be successful, especially for well-structured data. Along this direction, we will perform comprehensive experiments on the DT-Fixup algorithm which is designed to mitigate mentioned issues. For possible performance improvement on DT-Fixup, we propose to study the applicability of the power normalization and Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization (MADGRAD) in this setting. This is motivated by the recent literature which shows that, stemming from batch normalization widely adopted in the area of computer vision, power normalization is shown to outperform the layer normalization usually found in the transformers. In the family of AdaGrad adaptive gradient methods, MADGRAD is a new optimization technique that performs exceptionally well on deep learning optimization problems from a variety of fields, including classification and image-to-image tasks in vision and recurrent and bidirectionally-masked models in natural language processing. Even on issues where adaptive methods typically perform badly, MADGRAD matches or beats both SGD and ADAM in test set performance for each of these tasks. This research will be performed on ReClor, and LogiQA datasets selected according to its structure.

DEDICATION

I would like to dedicate this thesis to my family for her incredible love and support. Because I believe that she is the real backbone of our family, this is to appreciate her selfless hard work and efforts towards the family.

Furthermore, I dedicate it to my dad to raise me like a son and give me wings to fly. To my grandfather, for always trusting me and supporting me in my hard times, without his encouragement, nothing would have been easy. And to my entire family for their unconditional affection toward me.

ACKNOWLEDGEMENTS

I would like to sincerely express my most profound gratitude towards my supervisor Dr.Jessica Chen, whose input helped me immensely. With her input, I was able to look at my research with a different perspective and a more critical eye.

Secondly, I would like to express my gratitude to my thesis committee members for their beneficial advice and suggestions for my thesis.

I humbly extend my thanks to the School of Computer Science and all concerned people who helped me in this regard.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

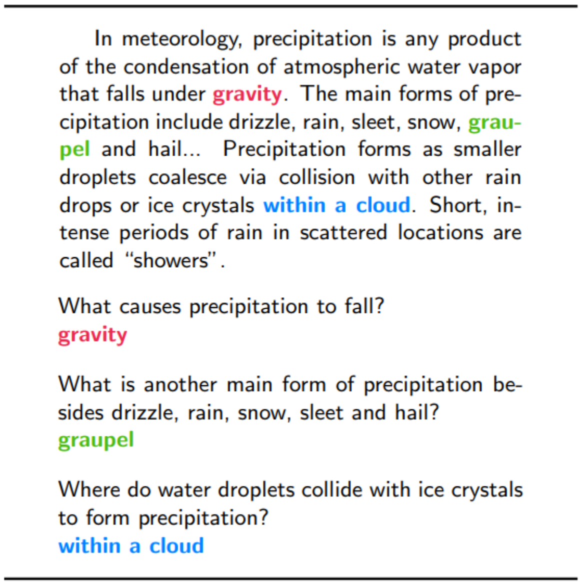
TP	True Positive
FN	False Negative
FP	False Positive
TN	True Negative
TPR	True Positive Rate
FPR	False Positive Rate
FNR	False Negative Rate
TNR	True Negative Rate
FRR	False Rejection Rate
FNMR	False Non Match Rate
FAR	False Rejection Rate
FMR	False Match Rate
EER	Equal Error Rate
SVM	Support Vector Machine
DT	Decision Tree
KNN	K Nearest Neighbors
NB	Naive Bayes
LR	Logistic Regression
RF	Random Forest
MLP	Multi Layer Perceptron
LGB	Light Gradient Boosting Machines

NN	Neural Network
GA-KNN	Genetic Algorithm -K Nearest Neighbors
IForest	IsolationForest

CHAPTER 1

Introduction

Reading comprehension (RC) is the capacity to read a piece of text and answer questions about it. This can be quite tough for machines as it demands an understanding of language as well as worldly knowledge. Take, for example, the question ”What causes precipitation to fall?” linked to a specific passage (referenced as Figure 1.0.1). To find the answer, one could first identify the pertinent section of the passage, which states ”Precipitation... falls under gravity.” Then, by understanding that ”under” is indicating a cause rather than a location, the correct response can be concluded: ”gravity.” This task, when performed by machines, forms the basis of Machine Reading Comprehension (MRC). [36]



In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”.

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Fig. 1.0.1: Question-answer pair from SQuAD dataset [36]

Machine Reading Comprehension (MRC) is a significant aspect of Natural Language Processing (NLP) research. The objective of this task is to design models that can interpret a provided piece of text and understand the responses to particular questions, which are related to the text's context. In recent years, the field of MRC has seen significant advancements, particularly due to the success of transformer(a machine learning model) [48]. Models like RoBERTa[24] that utilize pre-trained transformers have demonstrated impressive performance, nearly saturating most of the popular MRC datasets [36, 20, 35]. Transformers, introduced in the paper "Attention is All You Need" [48], are deep learning models with self-attention mechanisms. They excel in Natural Language Processing tasks, including Machine Reading Comprehension, by effectively grasping textual context. A prime example is the RoBERTa model [24], which we'll cover in Chapter 2 along with transformers [48].

This progress led researchers to consider other complexities in the field of machine reading comprehension. One such critical aspect is logical reasoning - the capacity to inspect, analyze, and critically assess arguments as they appear in everyday language, based on the definition provided by the Law School Admission Council[6]. Logical reasoning is a major component of human intelligence and is vital in various tasks such as negotiation, debating, and writing. However, current reading comprehension datasets lack or have very little data that necessitate logical reasoning. For example, the MCTest dataset [38] doesn't require logical reasoning at all, and only 1.2% of the SQuAD dataset [36] requires it, as noted by researchers Sugawara and Aizawa [43]. In order to further evolve models' capabilities in logical reasoning, from simple relationship classification to more complex reasoning, and from sentence-level to passage-level, there's a clear need for a reading comprehension dataset that emphasizes logical reasoning.

Let's look at an example of logical reasoning questions from the ReClor dataset [56], as shown in Figure 1.0.2. This format is similar to multiple-choice reading comprehension datasets. It includes a context, a question, and four potential answers, but only one is correct. To find the right answer, readers have to understand the logical links between the lines to spot the conflict, and then assess each of the options to select the one that resolves the conflict. Datasets like ReClor [56] and LogiQA [22] have been developed to encourage advancements in logical reasoning within Machine Reading Comprehension (MRC).

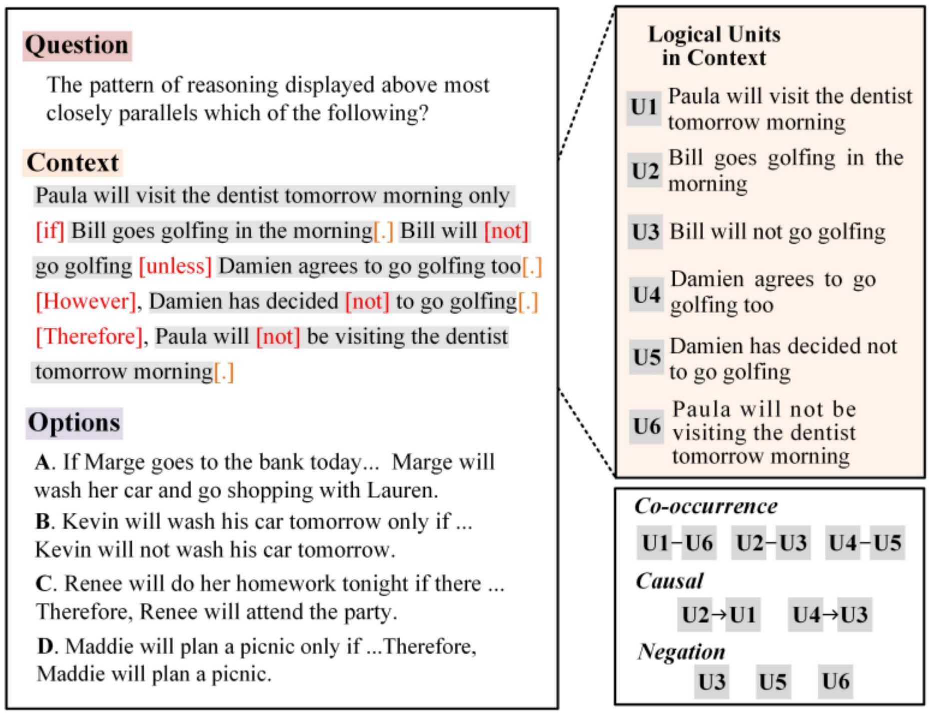

Question
The pattern of reasoning displayed above most closely parallels which of the following?
Context
Paula will visit the dentist tomorrow morning only [if] Bill goes golfing in the morning[.] Bill will [not] go golfing [unless] Damien agrees to go golfing too[.] [However], Damien has decided [not] to go golfing[.] [Therefore], Paula will [not] be visiting the dentist tomorrow morning[.]
Options
A. If Marge goes to the bank today... Marge will wash her car and go shopping with Lauren.
B. Kevin will wash his car tomorrow only if ... Kevin will not wash his car tomorrow.
C. Renee will do her homework tonight if there ... Therefore, Renee will attend the party.
D. Maddie will plan a picnic only if ...Therefore, Maddie will plan a picnic.
Logical Units in Context
U1 Paula will visit the dentist tomorrow morning
U2 Bill goes golfing in the morning
U3 Bill will not go golfing
U4 Damien agrees to go golfing too
U5 Damien has decided not to go golfing
U6 Paula will not be visiting the dentist tomorrow morning
Co-occurrence
U1–U6 U2–U3 U4–U5
Causal
U2→U1 U4→U3
Negation
U3 U5 U6

Fig. 1.0.2: Question-answer pair from ReClor dataset [56]

To tackle these logical reasoning challenges within MRC, several adaptations of the transformer models to enhance its capacity for logical reasoning, with some of the most notable ones being Focal Reasoner [28], MERIt [17], LReasoner[51], and DT-Fixup[54]. Our study primarily extends the work on DT-Fixup. Unlike Focal Reasoner, which introduces a new layer type in RoBERTa[24], or MERIt, which forms meta paths among logical variables, DT-Fixup addresses the initial optimization challenges that prevent a standard transformer layer from learning logical structures. It's one of the few approaches that directly confront the optimization problems in the transformer layer that impede the training of logical structures. We explore more about DT-Fixup in Chapter 3.

DT-Fixup [54] asserts that the primary optimization challenge stems from the

Adam optimizer and Layer Normalization, two concepts we'll delve into in chapters 2 and 3. The solution proposed by DT-Fixup involves adding extra transformer layers on top of pre-trained models to tackle the optimization issue. They achieve this by eliminating Layer Normalization and adjusting the weight initialization[54].

We, however, suggest Power Normalization[42] as a replacement for Layer Normalization since it maintains first-order smoothness, which aids in better model training. We also propose the use of MADGRAD[7] instead of ADAM[19], as MADGRAD has demonstrated good performance with sparse data points and has outperformed ADAM in numerous leading-edge problems. We will be discussing more about this in chapters 2 and 3.

The influences of Power Normalization [42] and MADGRAD [7] have not yet been thoroughly investigated in the specialized domain of Natural Language Processing (NLP), especially when it comes to training transformer models on smaller datasets. This is the primary motivation for our thesis; we aim to comprehend how PowerNorm and MADGRAD affect the DT-Fixup algorithm [54], as these insights could potentially enhance the performance of existing transformers.

CHAPTER 2

Background

This chapter will be presented to explain some of the fundamental concepts that have been used to understand the DT-Fixup methodology and how we conducted our investigation on it.

2.1 Transformers

The Transformer model was developed to overcome some of the drawbacks of earlier sequence-to-sequence models including recurrent neural networks (RNNs) [41] and convolutional neural networks[21] (CNNs). It was first presented in the publication ”Attention Is All You Need” by Vaswani et al. in 2017. The self-attention mechanism, which the Transformer model introduced, enabling it to attend to different parts of the input sequence at different times without being constrained by the sequential structure of RNNs or the fixed-size receptive fields of CNNs, was the model's key innovation. Self-attention was a concept that was not wholly new because it has been utilized in the area of natural language processing (NLP) for activities like language modelling and machine translation. However, prior models frequently merged RNNs or CNNs with self-attention, which constrained their scalability and increased their computational cost.

The Transformer model, on the other hand, was more effective and could handle longer sequences since it only used self-attention as a mechanism for attending to input sequences. The Transformer also made use of a multi-head attention mechanism that allowed it to focus on many aspects of the input sequence at once, significantly enhancing its performance. This is one of the reasons why transformers are better at logic compared to RNNs or CNNs.

2.1.0.1 Understanding Encoders

We have an encoder and a decoder layer which make up the Sequence to Sequence mode. Each component of the input sequence is processed by the encoder, which then condenses the resulting data into a single vector known as the context vector.

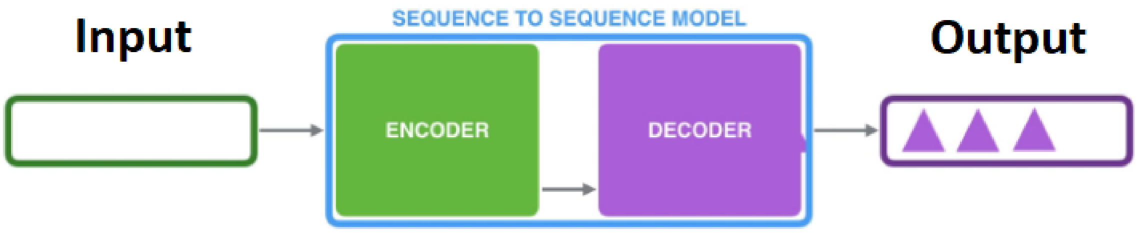


Fig. 2.1.1: Encoder and Decoder

The input words are first encoded using positional encoding. To encode the positioning information of words in a phrase, positional encoding is a technique used in the field of natural language processing (NLP). A sentence can be represented in NLP as a series of vectors, each of which represents a word in the sentence. It is crucial to convey information about each word's place in the sentence since the sequence of the words in a sentence matters. Positional encoding is useful in this situation. By using positional encoding, each word's matching vector representation can now include information about the word's location inside the phrase. This is accomplished by supplementing the word's initial embedding with a fixed-length vector known as positional encoding.

A set of hidden representations are created by the encoder from a sequence of input tokens (often words or subwords), which are then used by the decoder to create the output sequence. The encoder's job is to take the input sequence's significant information and express it in a way that is appropriate for subsequent operations. The encoder is made up of a stack of identical layers, each of which has a feedforward network and a self-attention mechanism as its two sublayers.

Before encoding a specific item, the encoder's inputs undergo a self-attention layer, enabling them to take into account other items within the input sequence. Subsequently, the self-attention layer's output is processed through a feed-forward neural network, which is applied individually to each point in the input sequence. As illustrated in Figure 2.1.2, the input vector (x1,x2,x3) is passed through the self-attention layer, resulting in output vectors z1, z2, and z3. These resultant vectors are then forwarded to the feed-forward network as previously mentioned.

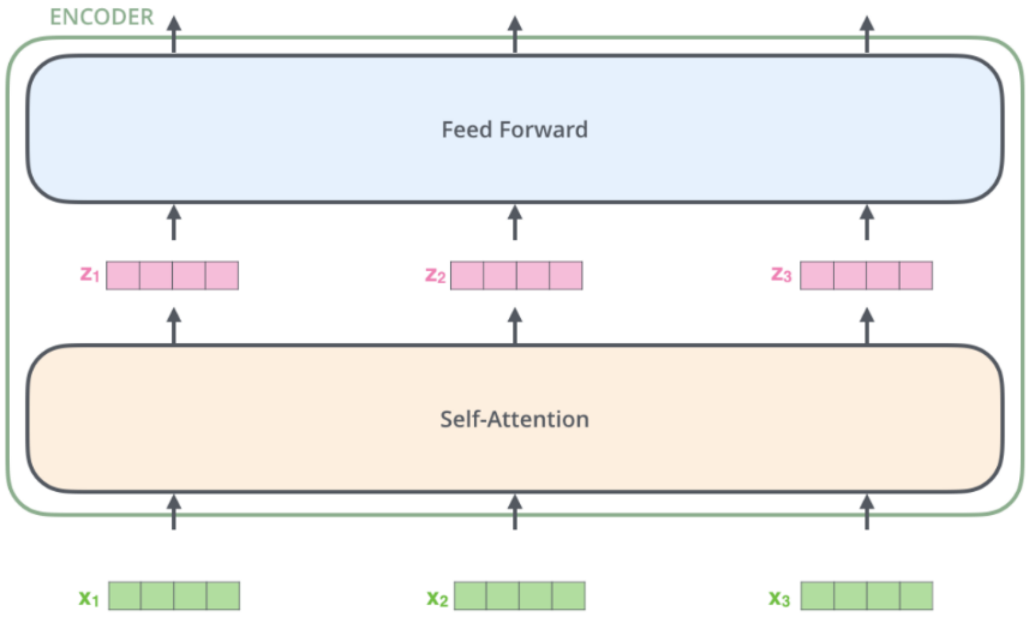


Fig. 2.1.2: Encoder Architecture

2.1.0.2 The Residuals

The structure of a Transformer's encoder includes repetitive units, each of which contains two parts: a self-attention mechanism and a position-wise fully connected feed-forward network. These parts are known as sub-layers. An important aspect of this architecture is that each sub-layer is surrounded by a residual connection, which directly combines the input and output of the sub-layer. This is designed to help with the issue of disappearing gradients when training deep networks. Furthermore, after every sub-layer, there is a step known as layer normalization which helps in stabilizing the learning process and reducing training time.

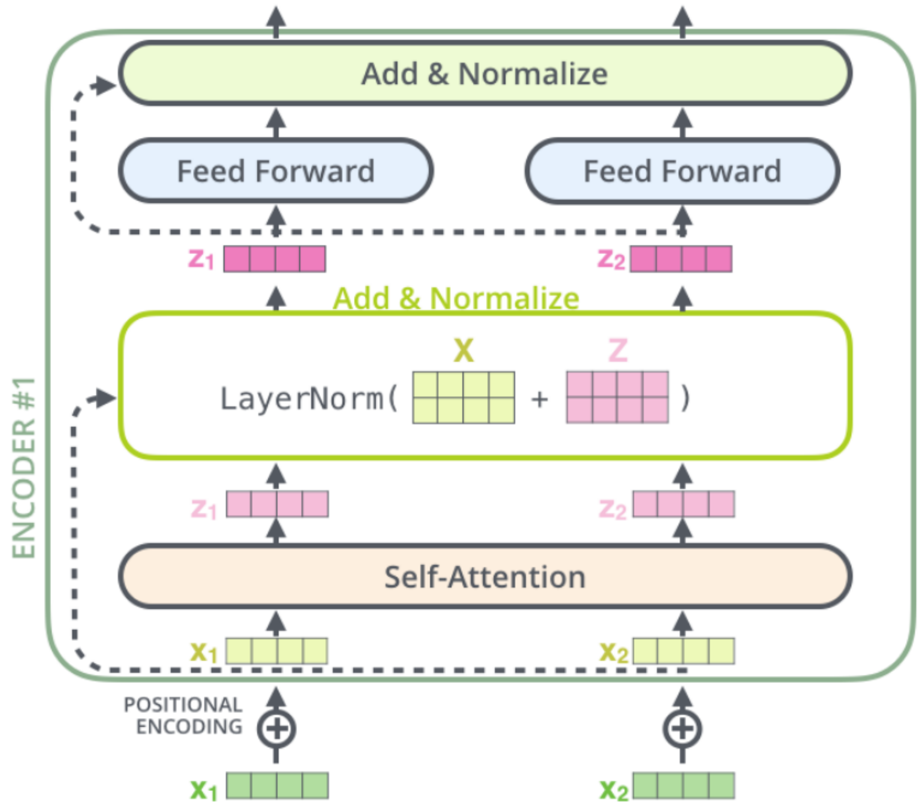


Fig. 2.1.3: Graphical Representation of Vector Interactions and Layer Normalization Process

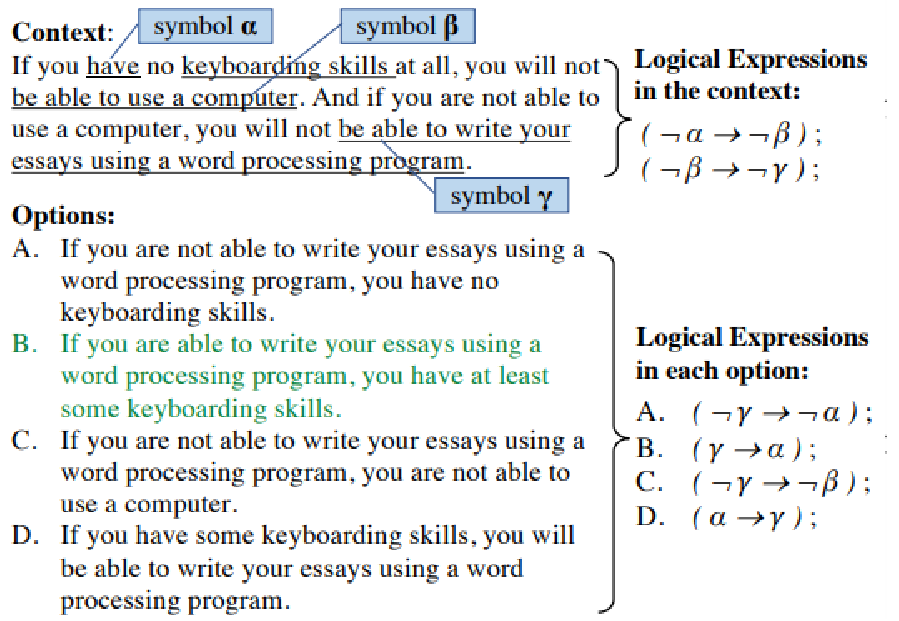
A residual connection, or a skip connection, helps in preventing the vanishing gradient problem during training and allows the model to learn more complex functions. It does this by adding the original input back to the output of the sub-layer. The output of each sub-layer is LayerNorm(x + Sublayer(x)), where x is the input to the sub-layer, and Sublayer(x) represents the function that the sub-layer itself implements.

We will be modifying the "Add & Normalize" layer for our work in this thesis.

2.2 Logical Reasoning

Logic theories delve into the symbolic reasoning processes used in everyday language. They can be broadly divided into informal logic and formal logic. Informal logic reveals the reasoning structure within context, whereas formal logic transforms the language into symbolic axiomatic systems to assess its validity. Both forms of logic contribute to the development of models for logical reasoning question-answering.

In the domain of logical reasoning, the first critical step is to discern and understand the elementary components that make up the reasoning process. We will be exploring that in the below sub-sections.



Context:
If you have no keyboarding skills at all, you will not be able to use a computer. And if you are not able to use a computer, you will not be able to write your essays using a word processing program.

Logical Expressions in the context:
$(\neg\alpha \rightarrow \neg\beta)$;
$(\neg\beta \rightarrow \neg\gamma)$;

Options:
A. If you are not able to write your essays using a word processing program, you have no keyboarding skills.
B. If you are able to write your essays using a word processing program, you have at least some keyboarding skills.
C. If you are not able to write your essays using a word processing program, you are not able to use a computer.
D. If you have some keyboarding skills, you will be able to write your essays using a word processing program.

Logical Expressions in each option:
A. $(\neg\gamma \rightarrow \neg\alpha)$;
B. $(\gamma \rightarrow \alpha)$;
C. $(\neg\gamma \rightarrow \neg\beta)$;
D. $(\alpha \rightarrow \gamma)$;

Fig. 2.2.1: Example from ReClor to understand Logical Reasoning[56]

2.2.1 Logic Identification

When it comes to logical reasoning, the initial crucial step involves identifying and comprehending the basic elements that constitute the reasoning process - these elements are what we call logical expressions. These logical expressions, comprised of logical symbols and connectives, can be found in each sentence of the text and every provided option. In the following section, we will delve into the methodology used to identify these logical expressions, including the use of notations for logical symbols and connectives, and the process of combining them. Furthermore, we will discuss how to detect the presence of negation and conditional relationships within these expressions, essential aspects in understanding and performing logical reasoning.

To perform logical reasoning, it's necessary to first recognize the fundamental elements that facilitate reasoning - we call these logical expressions. We spot these expressions in each context sentence and the corresponding options. To demonstrate the structure of these expressions, we use specific notations:

1. $\{\alpha, \beta, \gamma, \ldots\}$: These are the logical symbols that make up the fundamental elements in the context, helping to form the logical expressions. For instance, 'have keyboarding skills' in Figure 2 is such a symbol.

2. $\{\neg, \rightarrow\}$: This is the set of logical connectors. $\neg$ signifies a negation operation applied to a specific logical symbol, and $\rightarrow$ denotes a conditional relationship between two logical symbols.

3. $\{(\alpha \rightarrow \beta), \ldots\}$: These are the logical expressions made up of logical symbols and connectors. $(\alpha \rightarrow \beta)$ implies that α is a precondition for β.

If a negative word is associated with a logical symbol, denoted as α, we append the negation connector $\neg$ in front of α to form a new logical symbol $\neg\alpha$. We categorize words like "not", "n't", "unable", "no", "few", "little", "neither", "none of" as negative words. When there's a conditional relationship between two logical symbols, α and β, in a sentence, we can build the associated logical expression as $(\alpha \rightarrow \beta)$. The conditional relationship between the symbol α and β, denoted as $(\alpha \rightarrow \beta)$, is recognized based on conditional indicators like "if α, then β", "α in order for β", "β due to α", "$\neg\beta$ unless α", and so forth. In instances where an active voice is present between α and β, we also apply $(\alpha \rightarrow \beta)$. As demonstrated in Figure 2.2.1, given a context with two sentences, we can extract three logical symbols $\{\alpha,\ \beta,\ \gamma\}$ and recognize two existing logical expressions: $(\neg\alpha \rightarrow \neg\beta)$ and $(\neg\beta \rightarrow \neg\gamma)$.

2.2.2 Informal Logic

Logical Components in Arguments: Informal logic looks into the structural reasoning processes within arguments. This structure is referred to as an argument.

For example, in the argument "A and B; therefore C," "A," "B," and "C" are propositions, with "C" being a conclusion derived from the two premises "A" and

"B." As a result, within this discrete structure, the conclusion and premise are two key logical components, which are typically complete sentences or sub-sentences.

Inference Indicators: Informal logic aids in the process of identifying logical components from text and reconstructing the argument's structure. This is accomplished by using regularly occurring indicators that signal the premise or conclusion. Typical premise indicators include words like "since," "because," "for," and "given that," while conclusion indicators include "therefore," "so," and "consequently," among others. Drawing from this, we rebuild logical structures for logical reasoning questions and answers by employing these inference indicators as text delimiters. These delimiters split the text into multiple sentences or clauses, which ideally serve as the fundamental units for reasoning. The indicators themselves represent the respective logical relations between these units.

2.2.3 Formal Logic

Deviation of Logical Expressions: Variation in Logical Expressions. Formal logic systems like First-Order Logic (FOL) allow the derivation of numerous valid formulas, i.e., logical expressions, from a set of axioms and rules. The soundness of these derivations ensures that if the axioms are true, then the derived expressions will also hold true.

For instance, the rule of modus ponens in first-order propositional logic can be represented as:

$$P \to Q, P \vdash Q.$$

This suggests that if $\alpha \wedge \beta \to \gamma$ is an axiom and holds true, and if $\alpha \wedge \beta$ is also true, then we can conclude that γ is also true.

Another example is provided by the rule of addition: $P \vdash P \vee Q$. If we assume $\alpha \to \beta$ to be an axiom and to be true, then the derived expression $(\alpha \to \beta) \vee \gamma$ will also hold true.

From this, we can observe that in the derivation of logical expressions, the expressions derived from each other are related only if they share common variables. In the first example, this common variable is $\alpha \wedge \beta$, while in the second example, it is $\alpha \rightarrow \beta$. This observation has inspired us to create the variable edges during the construction of the logic graph.

Validity of Expressions and Instantiation: If a logical expression is valid, multiple instantiations based on it are also true as they follow the same valid reasoning pattern. For instance, consider two applications of the modus ponens rule from the equation above:

Example 1 is the classic syllogism: 'All men are mortal. Socrates is a man. Therefore, Socrates is mortal.' This is derived by substituting 'be men' for P and 'be mortal' for Q.

Example 2 follows the same reasoning: 'All birds can fly. Eagles are birds. Therefore, eagles can fly.' Here, 'be bird' is substituted for P, and 'can fly' for Q.

Even though the topics of Examples 1 and 2 are different, we know both statements are true because they share the valid reasoning framework of the modus ponens rule. In addition, logical reasoning processes in texts often use natural language, with logical variables embedded within the language. One clue to these logical variables can be found in the topic-related terms that often recur in the text, such as 'men' and 'mortal' in Example 1, and 'birds' and 'fly' in Example 2.

2.2.4 Examples of Logical Structures

- Necessary Assumptions: Determine the statement that must be accurate or is necessary for the reasoning to be valid. Let's denote the claim as P, which represents the statement that must be true or required for the argument to be valid.

  In formal logic, we can express the claim as follows:

$$P \rightarrow \text{(Argument)}$$

  Here, the arrow ($\rightarrow$) represents the implication or logical consequence. The statement asserts that if claim P is true, then the argument as a whole holds or is valid.

Please note that this representation assumes a binary logic framework, where P is either true or false. In a formal logical analysis, additional premises or conditions may be included, and the logical structure may vary depending on the specific argument being evaluated.

- Sufficient Assumptions: Determine a comprehensive supposition that, when included in the reasoning, would make it logically sound. Determine a comprehensive supposition that, when included in the reasoning, would make it logically sound.

  In formal logic, we can express the supposition as follows:

  $$Q \rightarrow (\text{Argument})$$

  Here, the arrow ($\rightarrow$) represents the implication or logical consequence. The statement asserts that if the supposition Q is true, then the argument as a whole holds or is valid.

- Weaken: Determine details that could potentially undermine a given argument. Determine details that could potentially undermine a given argument.

  In formal logic, we can express the potential undermining details as follows:

  $$R \rightarrow \neg(\text{Argument})$$

  Here, the arrow ($\rightarrow$) represents the implication, and the negation symbol ($\neg$) represents the logical negation or "not" operator. The statement asserts that if the details R are true, then it is not the case that the argument holds or is valid.

- Identify a Flaw: Determine a mistake or error in the logic of an argument. Determine a mistake or error in the logic of an argument.

In formal logic, we can express the mistake or error as follows:

$$M \to \neg(\text{Argument})$$

Here, the arrow ($\to$) represents the implication, and the negation symbol ($\neg$) represents the logical negation or "not" operator. The statement asserts that if the mistake or error M is true, then it is not the case that the argument holds or is valid.

- Universal Affirmative (All/Everyone/Any):

$$\text{Example: All humans are mortal.} \quad (\forall x)\,\text{Human}(x) \to \text{Mortal}(x)$$

- Universal Negative (No):

$$\text{Example: No cats can fly.} \quad (\forall x)\,\text{Cat}(x) \to \neg\text{CanFly}(x)$$

- Particular Affirmative (Some):

$$\text{Example: Some birds can sing.} \quad (\exists x)\,\text{Bird}(x) \wedge \text{CanSing}(x)$$

- Sufficient conditional reasoning: uses conditional statements in the form "If P, then Q" to reason hypothetically. "If P, then Q" general form: If P, then Q. $\quad P \to Q$

$$\text{Example: If it rains, then the ground is wet.} \quad \text{Rain} \to \text{Ground is wet}$$

- Necessary conditional reasoning: uses conditional statements like "P only if Q" and "Q whenever P" to establish necessary conditions for P.

$$\text{"P only if Q" general form: P only if Q.} \quad P \Rightarrow Q$$

$$\text{"Q whenever P" general form: Q whenever P.} \quad P \Rightarrow Q$$

- Disjunctive reasoning: uses disjunctive premises in the form ”either...or...”, where the conclusion is true if at least one premise is true.
  Disjunctive Reasoning Example:

  Premise 1: Either it is raining or the sun is shining. ($Raining \lor SunShining$) Premise 2: I am getting wet. (Wet) Conclusion: Therefore, it is raining. ($Raining$)

  In this example, the disjunctive premises state that it is either raining or the sun is shining. The conclusion is derived based on the observation that I am getting wet, leading to the inference that it is raining.

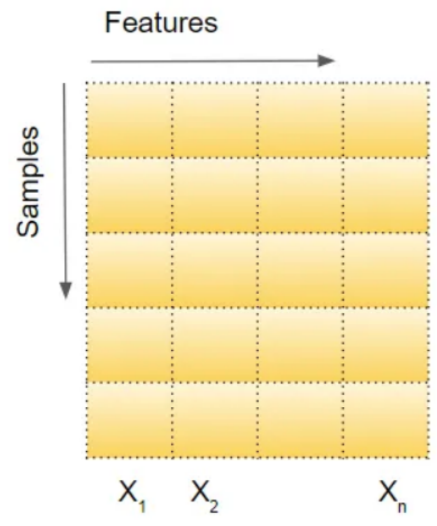
- Conjunctive reasoning: uses conjunctive premises in the form ”both...and...”, where the conclusion is true only if all premises are true. Conjunctive Reasoning Example:

  Premise 1: Both the sun is shining and the birds are singing. ($SunShining \land BirdsSinging$) Premise 2: The weather is pleasant. (PleasantWeather) Conclusion: Therefore, it is a beautiful day. ($BeautifulDay$)

  In this example, the conjunctive premises state that both the sun is shining and the birds are singing. The conclusion is derived based on the observation that the weather is pleasant, leading to the inference that it is a beautiful day.

2.3 Normalization

Normalization is a common practice in deep learning, where input data is adjusted to have a mean of zero and a variance of one. This process is applied to each feature column individually. In other words, if we have multiple features (x1, x2, ..., xn) with different value ranges, normalization ensures that each feature is transformed to have a similar scale as shown in Figure 2.3.1.



$$X_i = \frac{X_i - Mean_i}{StdDev_i}$$

Fig. 2.3.1: how normalization takes place

To achieve this, we calculate the mean and variance of each feature across all the samples in the dataset. Then, we use these computed statistics to normalize the values of each feature. The formula used for normalization typically involves subtracting the mean from each value and dividing the result by the standard deviation (square root of the variance).

The purpose of normalization is to bring the features onto a comparable scale, preventing any particular feature from dominating the learning process due to its larger magnitude. It helps the model to converge faster during training and ensures that the gradients are not influenced disproportionately by certain features. Normalizing the data can also help avoid numerical instability issues that may arise in the optimization process.

By normalizing the input data, we create a more balanced representation of the features, allowing the deep learning model to make fair and accurate comparisons between them.

In the provided image (Figure 2.3.2), we observe the impact of normalizing data. The original values, represented by the blue data points, have been transformed to be centred around zero, as indicated by the red data points. This process ensures that all the feature values are now on the same scale, making them comparable and facilitating fair analysis and interpretation. By aligning the data around zero, normalization eliminates any bias or influence that might arise due to varying scales of different features.

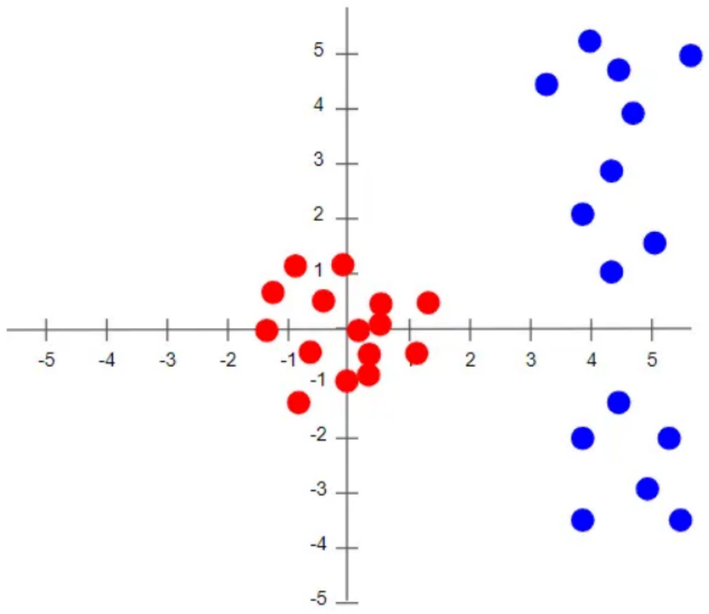


Fig. 2.3.2: What normalized data looks like

Let's look at an example (Figure 2.3.3) with just two features that are on vastly different scales to comprehend what occurs without normalization. The network learns weights for each feature that are also on various scales because the output is a linear combination of each feature vector. Otherwise, the small feature will be completely masked by the large feature.

The network would therefore need to significantly update one weight relative to the other weight during gradient descent in order to ”move the needle” for the Loss. Because of this, it might take more steps to reach the minimum as the gradient descent trajectory oscillates back and forth along one dimension.

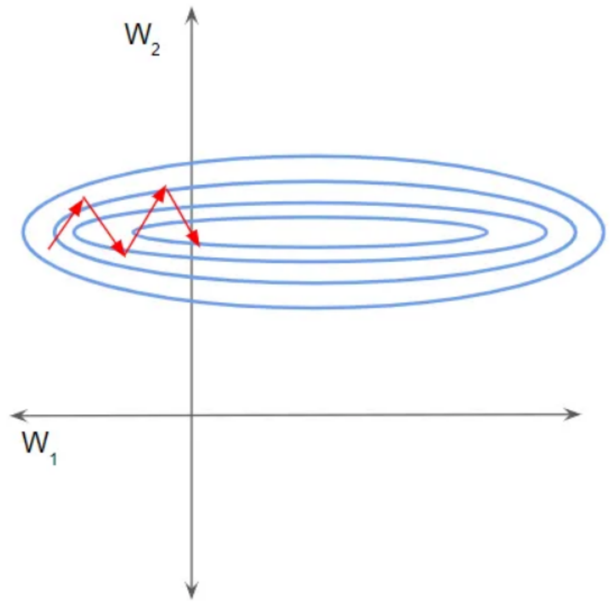


Fig. 2.3.3: Features on different scales take longer to reach the minimum

The loss landscape in this instance resembles a deep valley. The gradient can be divided into its two components. Along one dimension, it is very steep, and on the other, it is considerably milder as demonstrated in figure 2.3.4.

Due to one weight's significant gradient, we ultimately update it more significantly. The gradient descent is afterwards made to bounce to the opposite side of the slope. On the other hand, because of the second direction's smaller gradient, our weight updates and resulting steps are smaller. The network takes longer to converge on this unequal trajectory.

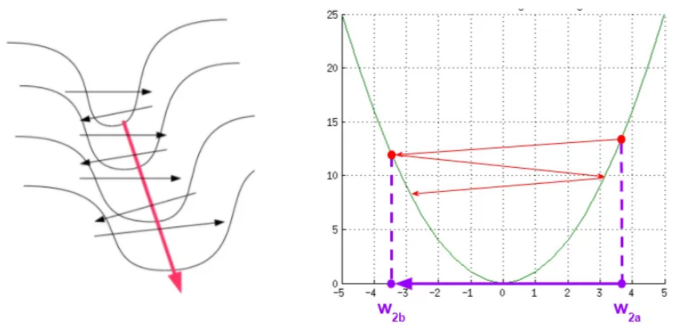


Fig. 2.3.4: A narrow valley causes gradient descent to bounce from one slope to the other

Instead, the lost landscape is more homogeneous like a bowl if the features are scaled similarly. Then, a smooth gradient drop can continue all the way to zero as shown in figure 2.3.5.

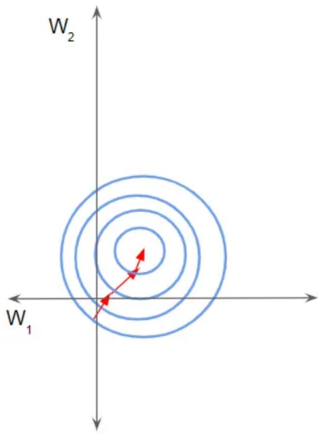


Fig. 2.3.5: Normalized data helps the network converge faster

2.4 Batch Normalization

When using a neural network with hidden layers, the output of one layer becomes the input for the next layer. If the inputs to a layer change dramatically, it can lead to unstable gradients, which can hinder the training process.

In the case of large datasets, they are usually divided into smaller batches for training using the mini-batch gradient descent algorithm. This algorithm optimizes the neural network's parameters by processing the dataset one batch at a time.

However, it's possible that the distribution of inputs to a specific layer changes across different batches. This change in distribution is known as internal covariate shift, as mentioned in the paper "Batch Normalization: Accelerating Deep Network Training" by Sergey Ioffe and Christian Szegedy [16]. For example, if the input distribution to layer K keeps changing across batches, it can result in a longer training time for the network.

During the mini-batch gradient descent algorithm, the weights and biases of the neural network are updated for each batch in the input dataset. The goal is to make the network fit the specific distribution of the input seen in that batch.

The problem arises when the distribution of the input changes significantly for the next batch. In such cases, the network has to adjust its parameters to fit the new distribution, which slows down the training process.

To overcome this issue, we can normalize the inputs to the hidden layers of the network. By doing so, we can mitigate the negative effects of large activations and changing distributions at the layer's input. This normalization is performed batch by batch, using a technique called batch normalization. Its purpose is to speed up the training process by ensuring that each mini-batch is normalized before updating the network's parameters.

For each hidden layer in a neural network, we apply a non-linear activation function to the inputs, which produces the layer's output. We have the ability to ensure that the pre-activations of every neuron in a specific layer have a mean of zero and a standard deviation of one. This can be achieved by subtracting the mean value from each input feature across the mini-batch and dividing it by the standard deviation.

To accomplish this, the authors of BatchNorm[16] introduce a layer following the output of the previous layer. This additional layer performs the normalization operation across the mini-batch, ensuring that the pre-activations at the current layer have a standard Gaussian distribution. The diagram in Figure 2.4.1 provides a visual representation of this process.

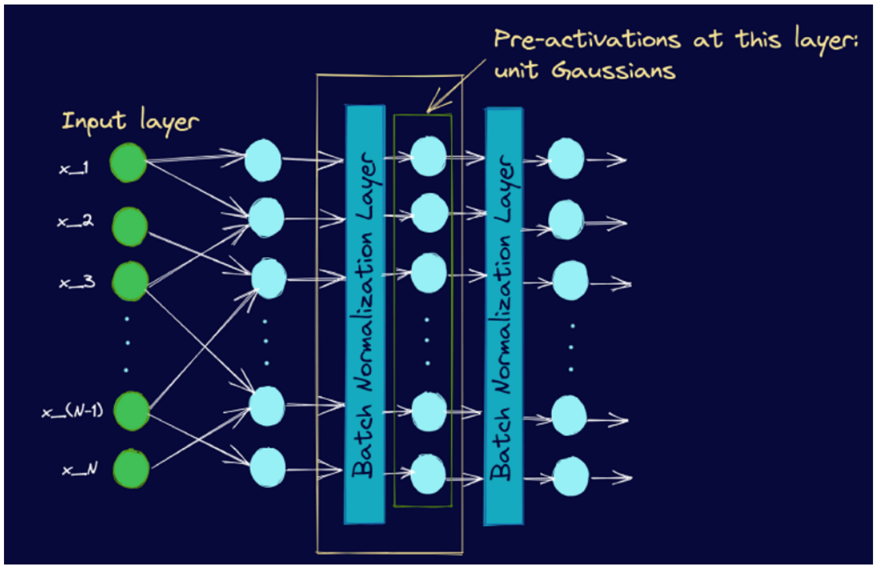

Pre-activations at this layer: unit Gaussians
Input layer
x_1
x_2
x_3
x_(N-1)
x_N
Batch Normalization Layer
Batch Normalization Layer

Fig. 2.4.1: Visual Representation of BatchNorm

Let's consider a mini-batch with 3 input samples, each input vector being four features long. The mean and standard deviation computed across each layer for each input as seen in figure 2.4.2.

By performing these calculations, we normalize the values based on the mean and standard deviation of the mini-batch, ensuring that the data is centred around zero and has a unit standard deviation across the layers.

2.5 LayerNorm

Layer Normalization[1], proposed by researchers Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton, ensures that all neurons within a particular layer of a neural network share the same distribution across all features for a given input.

In simpler terms, in Layer Normalization, they aim to make sure that each neuron in a layer sees similar values for each feature in the input.

For instance, if an input has d features, forming a d-dimensional vector, and there are B elements in a batch, the normalization is performed along the length of the d-dimensional vector, rather than considering the entire batch of size B.

This approach of normalizing across all features, but for each input individually, removes the reliance on batches. As a result, Layer Normalization is well-suited for sequence models like transformers and recurrent neural networks (RNNs) that were widely used before the advent of transformers.

To further illustrate, let's consider an example presented in Figure 2.5.1. Suppose we have a mini-batch containing three input samples, each having four features. In this case, we calculate the mean and variance for each feature separately, treating each input independently.

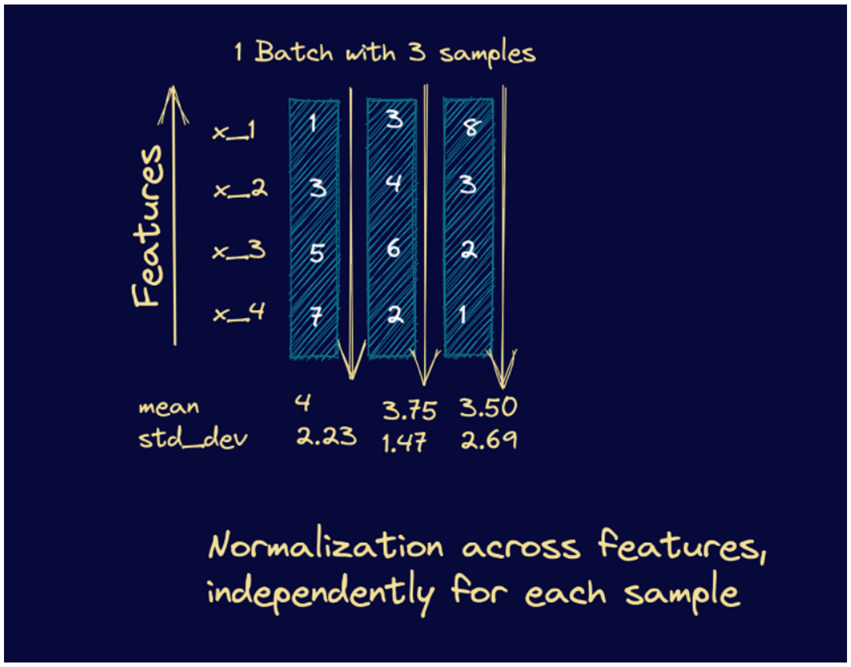


Fig. 2.5.1: Visual Representation of LayerNorm

The purpose of Layer Normalization is to ensure that each neuron within a layer receives consistent inputs across all features, enabling more stable and effective training of sequence models.

2.6 Power Normalization

Power Normalization [42] builds upon the concept of Batch Normalization [16], so a prerequisite for comprehending Power Normalization is understanding Batch Normalization. In NLP, LN (Layer Normalization)[1] is commonly preferred over BN

(Batch Normalization)[16] due to practical observations that a simple implementation of BN leads to poor performance in NLP tasks. The authors who developed the PowerNorm[42] method conducted a detailed investigation of the problems associated with BN in NLP. Based on their findings, they introduced a new normalization technique called Power Normalization (PN)[42], which surpasses LN in terms of performance[42].

The authors of the PowerNorm method[42] observed that the batch statistics of NLP data have a high variance throughout training, and this variance also exists in the corresponding gradients. To address this issue, they proposed a modification to BN called PN-V, which relaxes the zero-mean normalization (meaning they don't have to normalize exactly to zero) and replaces the variance with the quadratic mean to reduce the variation of batch statistics.

The authors of the Power Normalization paper [42] demonstrated that their modification preserves the first-order smoothness property of Batch Normalization, as supported by theoretical analysis.

In Batch Normalization, first-order smoothing is employed to maintain consistent mean and variance values across different mini-batches during the training process. This is achieved by calculating a running average of the mean and variance over all mini-batches. By reducing the variance in the normalization statistics, first-order smoothing helps in stabilizing the training process. [16]

The degree of smoothing in the first-order smoothing process is controlled by mathematical parameters that determine the extent of smoothing applied. By preserving first-order smoothness, models trained with Power Normalization exhibit smoother loss curves. This smoother behaviour aids in faster and easier convergence during training.

Figure 2.6.1 shows a visual representation of PowerNorm taking place. Their research [42], demonstrates that PN surpasses LN in terms of performance for neural machine translation and language modelling [36].

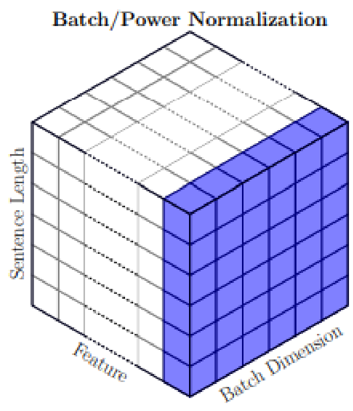


Fig. 2.6.1: Visual Representation of PowerNorm[42]

2.7 Optimization

Optimization algorithms are crucial for training neural networks as they help in finding the best set of model parameters, such as weights and biases, to improve the network's predictions. The most commonly used optimization technique is gradient descent. Deep learning libraries like PyTorch and Keras offer a variety of built-in optimizers based on gradient descent, including Adagrad [9] and Adam [19].

The reason for having multiple optimization algorithms is to cater to different problem types and data characteristics. Each algorithm has a specific formula for updating the model parameters, and understanding the significance of each formula is important in selecting the appropriate optimizer.

2.7.1 Loss Curve

The image depicts a neural network with two weight parameters. The horizontal plane represents the two axes corresponding to the weights, w1 and w2. The vertical axis represents the loss value associated with different combinations of weights. Essentially, the graph represents the ”Loss Landscape” of the neural network, illustrating the loss values for various weight values while keeping the input dataset constant.

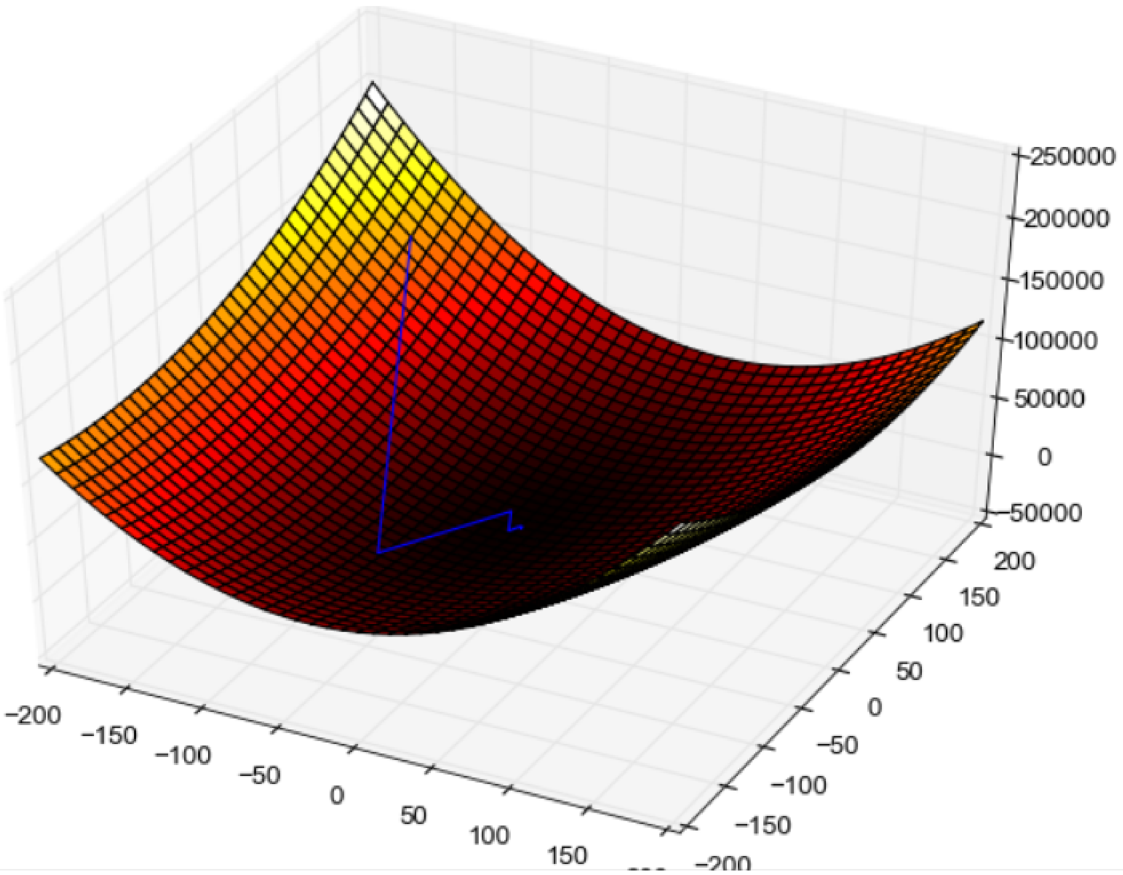


Fig. 2.7.1: Loss curve for Gradient Descent

The blue line on the graph in figure 2.7.1 traces the trajectory of the gradient descent algorithm during the optimization process. Initially, the algorithm randomly selects values for the weights and computes the corresponding loss value. In each iteration, it updates the weight values, aiming to reach a lower loss value along the curve. Eventually, it reaches its objective, which is the lowest point on the curve where the loss is minimized.

Loss curves in Figure 2.7.1 are helpful visualizations for comprehending gradient descent, but it's important to recognize that the depicted smooth, convex curve is an idealized representation rather than a realistic scenario.

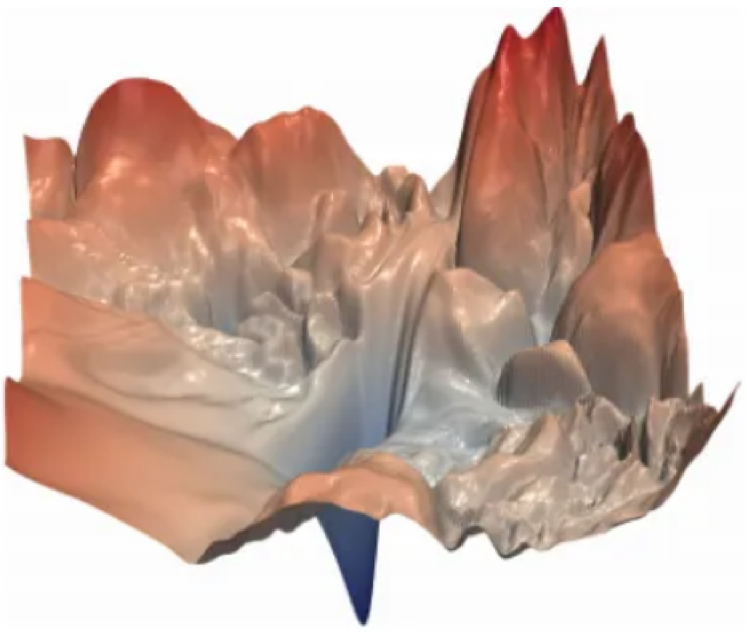


Fig. 2.7.2: Loss curve in CNN

In reality, the loss curve tends to be much more erratic and bumpy as shown in Figure 2.7.2. It may exhibit various irregularities and fluctuations instead of the smoothness and convexity shown in the picture. The actual loss landscape can have multiple local minima, saddle points, and other complex features that can pose challenges for optimization algorithms.

Furthermore, it is important to note that real-world neural networks typically have far more than just two parameters. In fact, they often consist of tens or even hundreds of millions of parameters. This vast number of parameters makes it impossible to visualize or conceptualize the entire parameter space in our minds. The complexity and high dimensionality of these networks pose unique challenges for optimization and require sophisticated techniques to effectively navigate and optimize the enormous parameter space.

2.7.2 Challenges with Gradient Descent Optimization

In a typical loss curve, there are often multiple local minima, along with the global minimum. Gradient Descent, which aims to descend towards lower loss values, faces a challenge when encountering local minima. Once the algorithm descends into a local minimum, it can struggle to climb back up the slope and may become trapped, unable to reach the global minimum.

This issue is known as being stuck in local minima. It hampers the optimization process, as the algorithm settles for suboptimal solutions instead of finding the global minimum, which represents the best parameter values for the given problem. Overcoming local minima is a crucial concern in optimization, and various techniques are employed to mitigate this problem and improve the chances of finding better solutions. An example is shown in Figure 2.7.3.

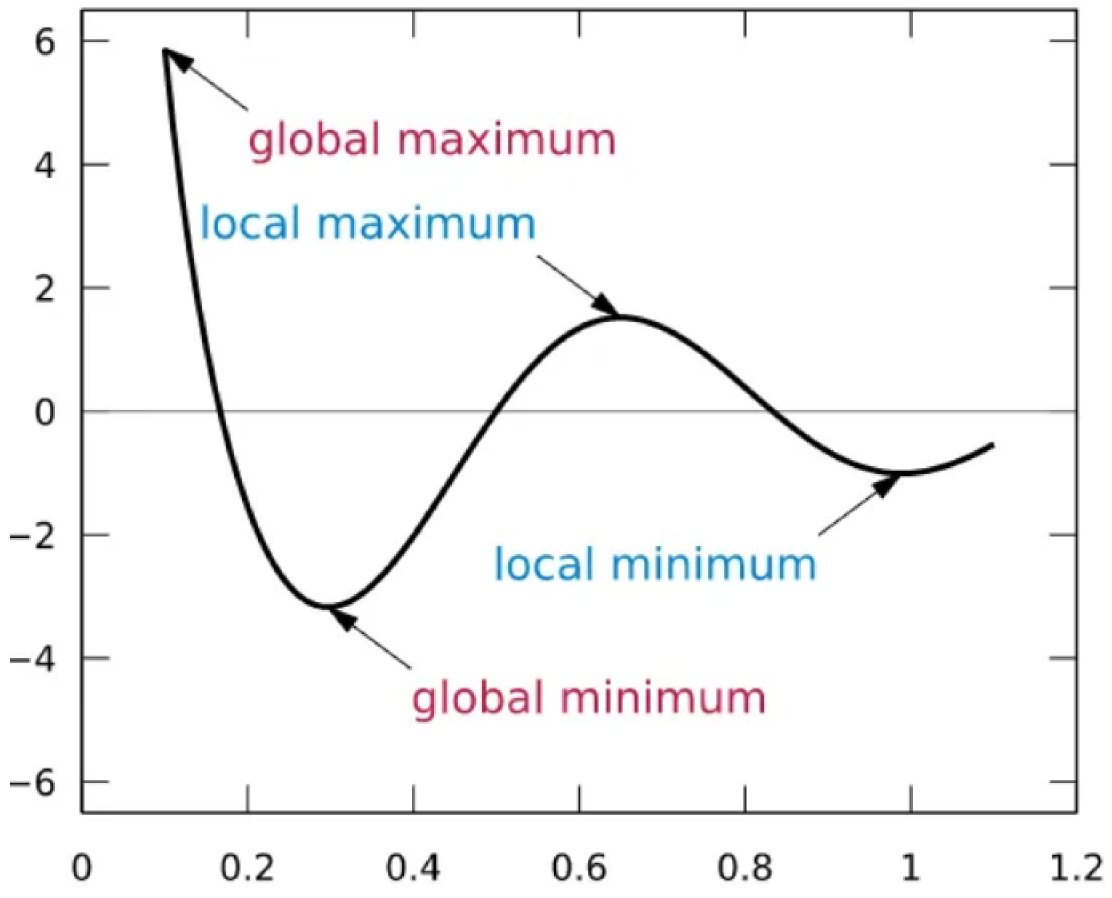


Fig. 2.7.3: local minimum

Another significant challenge in optimization arises from the presence of "saddle points." These are points in the loss curve where, in one direction corresponding to a particular parameter, the curve reaches a local minimum. However, in another direction corresponding to a different parameter, the curve reaches a local maximum.

Saddle points can hinder the progress of optimization algorithms because they exhibit a flat region where the gradient becomes close to zero in multiple dimensions. As a result, the algorithm may struggle to move past these points as the gradient provides ambiguous guidance.

While saddle points can be problematic, they are generally less common than local minima. Nonetheless, addressing their influence on optimization remains an important consideration in order to enhance the effectiveness and efficiency of the training process. An example is shown in Figure 2.7.4.

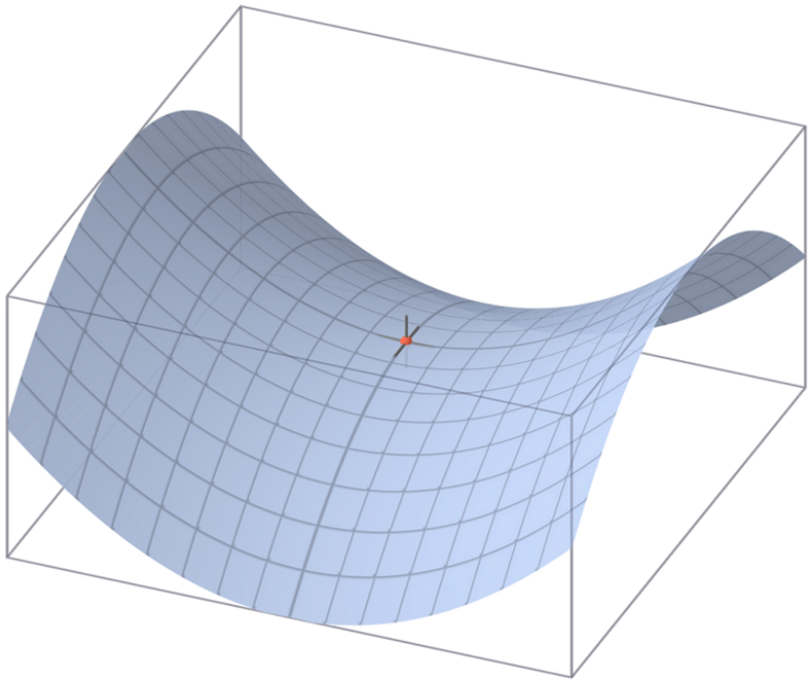


Fig. 2.7.4: Saddle point

Gradient Descent also faces challenges when traversing ravines, which refer to long and narrow valleys that slope steeply in one direction (the valley sides) and gently in another direction (along the valley). These ravines often lead down to the minimum of the loss curve. Due to the difficult navigation through these ravines, this shape is known as Pathological Curvature. You can imagine it as a narrow river valley that gently slopes down from the hills until it reaches a lake. The goal is to move swiftly downstream in the direction of the valley. However, Gradient Descent can easily oscillate back and forth along the sides of the valley, resulting in slow progress in the downstream direction.

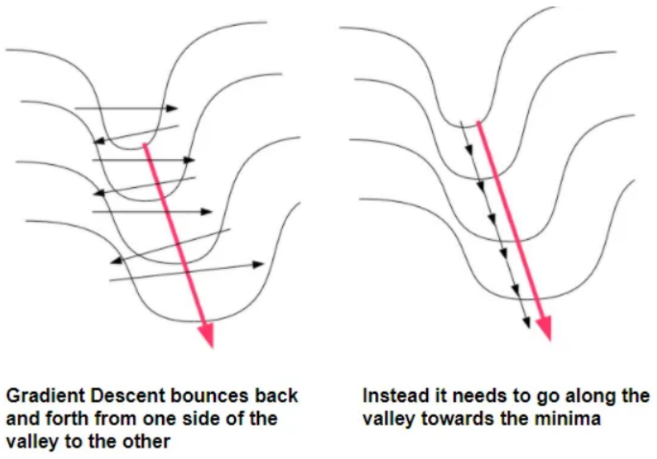

Gradient Descent bounces back and forth from one side of the valley to the other
Instead it needs to go along the valley towards the minima

Fig. 2.7.5: Valley

To address these challenges, optimization algorithms have evolved beyond vanilla Gradient Descent by incorporating various improvements. These enhancements aim to overcome the difficulties posed by local minima, saddle points, and ravines.

Let's explore the workings of batch gradient descent, an optimization technique, and how it progresses toward the minimum. In batch gradient descent, the algorithm calculates the gradient of the loss function for the entire training dataset in each epoch. However, this approach can be computationally expensive, especially for large datasets.

During training, batch gradient descent updates the model parameters based on the average gradient computed from all the training examples. It takes a lot of epochs, or iterations over the entire dataset, to reach the minimum of the loss function.

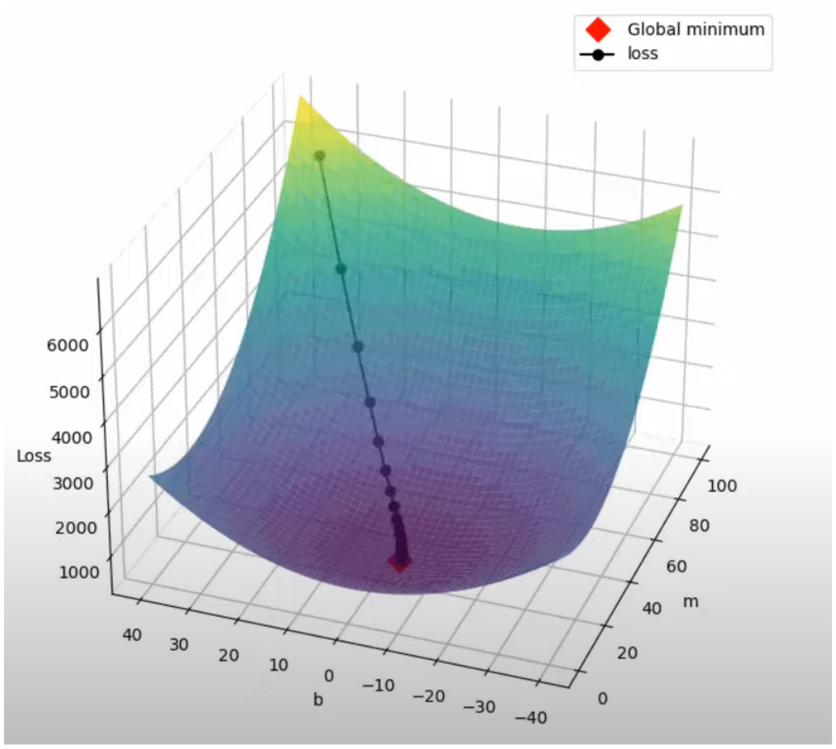


Fig. 2.7.6: Batch Gradient Descent

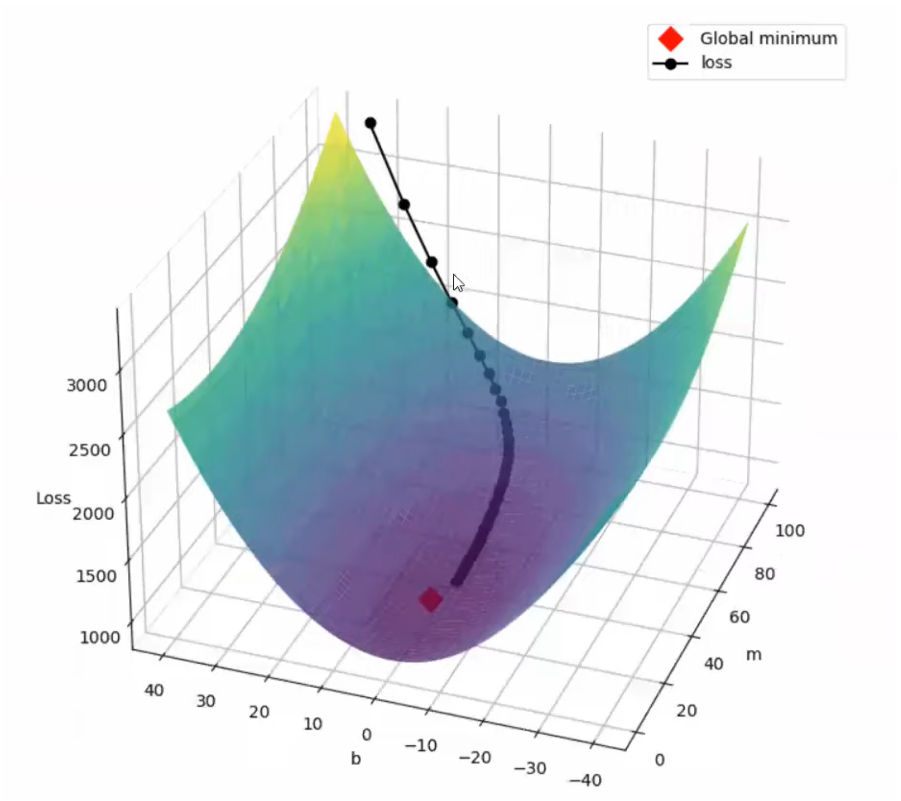


Fig. 2.7.7: Batch Gradient Descent in a Valley

An upgrade to batch gradient descent [31] involves the introduction of momentum, which aims to expedite convergence. Momentum is a technique where the update step considers both the current gradient and the accumulated gradient from previous iterations. By incorporating momentum, the algorithm gains momentum in the direction of steeper gradients, resulting in faster convergence.

However, one drawback of momentum is that it can lead to oscillations near local minima. The increased speed and momentum can cause the algorithm to overshoot the minimum and oscillate around it, slowing down the convergence process. This oscillation phenomenon can hinder the algorithm's ability to settle down and reach the precise minimum.

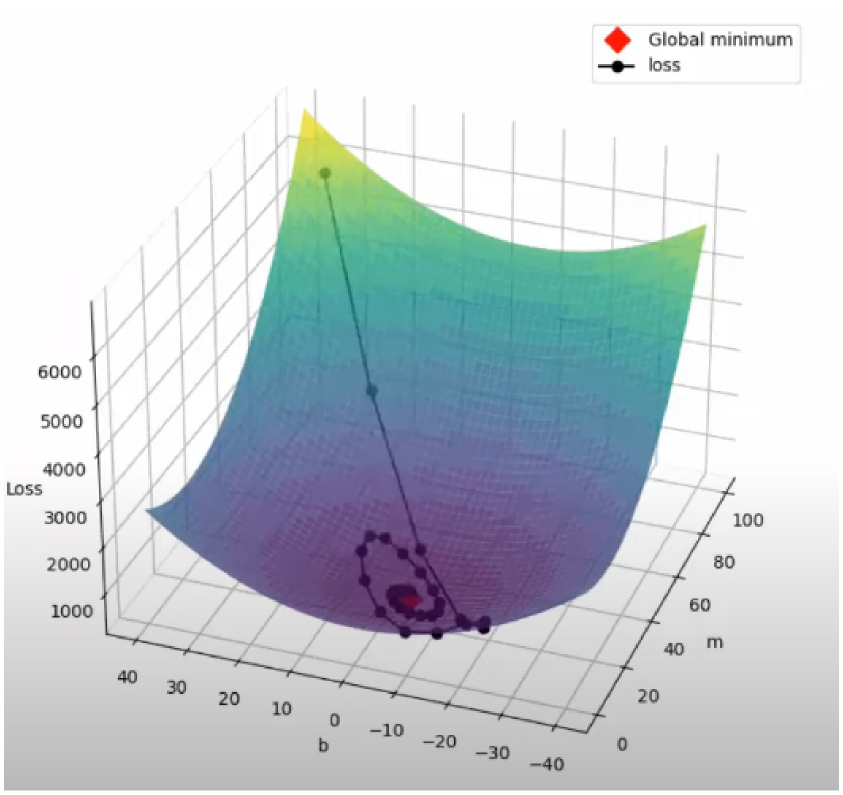


Fig. 2.7.8: Moment

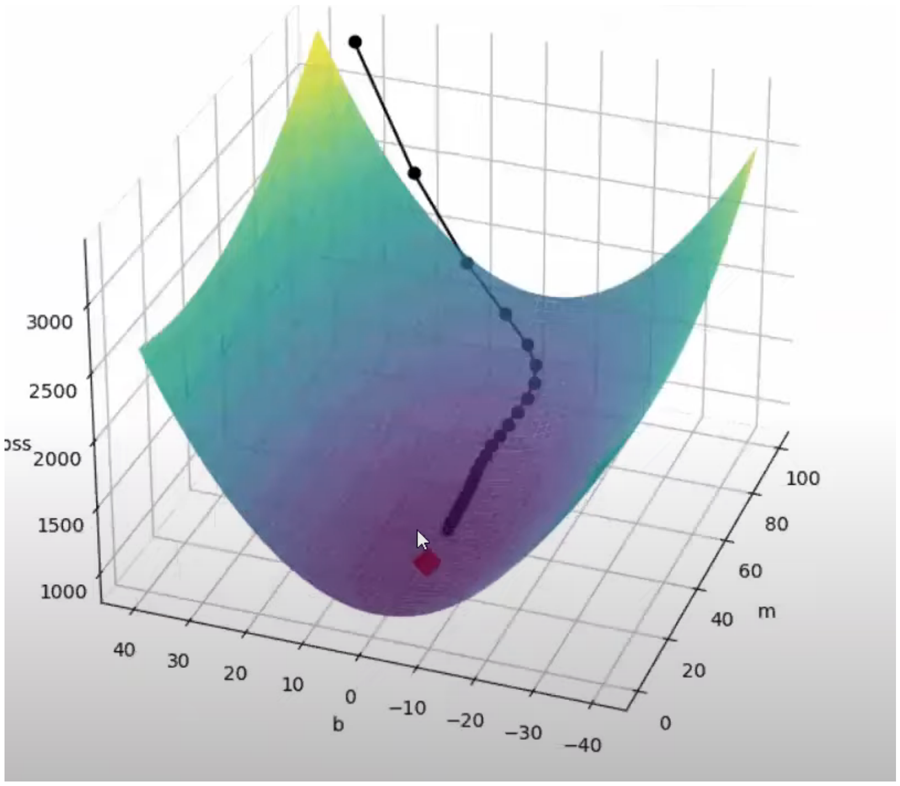


Fig. 2.7.9: Moment in a valley

Despite this drawback, momentum is generally beneficial in optimizing the training process by enabling faster convergence. However, momentum does not converge well when the loss curve resembles a valley. Valleys in the loss curve typically arise when the data is sparse, meaning there are regions in the feature space with limited or sparse data samples.

In such scenarios, the momentum-based optimization algorithms, including those using adaptive learning rates, may encounter challenges. The high momentum can cause the algorithm to overshoot the minimum and lead to oscillations within the valley. This behaviour can hinder convergence and make it difficult for the algorithm to navigate through the sparse regions effectively.

To address the challenges posed by valleys and sparse data, one effective technique is the use of AdaGrad, which stands for adaptive gradient. AdaGrad[9] is an optimization algorithm that adjusts the learning rate for each parameter based on the historical accumulation of gradients. It performs better in scenarios where the data is sparse.

By adapting the learning rate, AdaGrad places a greater emphasis on infrequent and important features in the data, enabling effective learning even in the presence of sparse data. This adaptive behaviour helps the algorithm converge more efficiently in regions with sparse data and navigate through valleys in the loss curve more effectively.

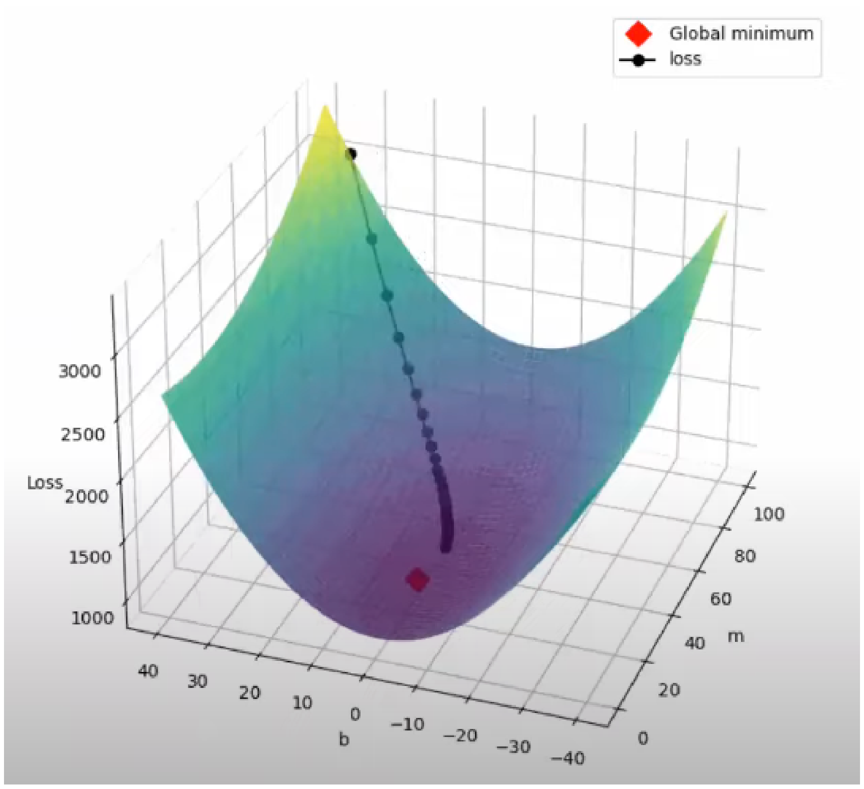


Fig. 2.7.10: Adagrad in a valley

AdaGrad's ability to adapt the learning rate for individual parameters makes it well-suited for handling sparse data, as it allows the algorithm to allocate more learning resources to relevant features while mitigating the impact of noisy or less informative features.

2.8 MADGRAD Optimization

The MADGRAD paper [7] discusses optimization for deep learning, which is a relatively new and growing sub-field in the optimization community. Unlike classical first-order optimization, deep learning problems involve additional concerns requiring new tools to overcome. Deep learning problems are characterized by very large parameter vector sizes, making it computationally infeasible to store matrices of the corresponding size. As a result, diagonal scaling approaches have become the industry standard for deep learning.

In the optimization community, deep learning optimization is a developing subfield that has more difficulties than traditional optimization. When opposed to traditional first-order optimization, deep learning problems pose particular difficulties that call for novel solutions. The size of the parameter vectors makes it impossible to store matrices of size DxD (where D is the dimension of the weight matrix), even with "limited memory" techniques [22,2]. Storage that is fixed at a small multiple of the parameter vector size is the practical limit on these issues.

Models with over 100 billion parameters are being investigated, but storage is only available in modest multiples of the size of the parameter vector. It is practically impossible to store (in RAM) matrices of size DxD for deep learning. Due to this, diagonal scaling techniques are now accepted as best practices for deep learning optimization. There are currently no other adaptive methods that consistently outperform Adam, making it the benchmark method in this class. [7]

Diagonal scaling methods, where adaptivity is carried out separately for each coordinate, are the industry standard for deep learning optimization. This results in memory use that scales as O (D). Adam relies on the illustrious past of diagonal adaptive techniques like RMSProp and AdaGrad.

The benchmark method for diagonal scaling approaches is Adam, which has seen widespread adoption. However, Adam has limitations, including underperformance in certain situations and a lack of convergence in some cases. To address these issues, the authors developed the MADGRAD method, which performs consistently at a state-of-the-art level across various deep-learning problems without requiring more tuning than Adam.

MADGRAD is based on the dual averaging formulation of AdaGrad, which has not been widely used for deep learning optimization despite having a simpler and more elegant theory compared to the mirror descent form. The authors argue that the lack of adoption of dual averaging approaches is due to misconceptions, including the belief that dual averaging is only interesting in the composite optimization setting and the issue of implementing dual averaging without considering the necessary modifications for deep learning.

In the case of Natural Language Processing (NLP), it is hypothesized that AdaGrad may converge faster compared to other optimization algorithms. This hypothesis stems from AdaGrad's inherent capability to handle sparse data and navigate valleys in the loss curve more effectively.

NLP tasks often involve dealing with sparse data, such as text data, where the feature space can be high-dimensional and data samples may be limited. In such scenarios, the loss curve may exhibit valley-like shapes due to the sparsity of the data.

Given AdaGrad's adaptive learning rate scheme, it is well-suited to handle sparse data. By appropriately adjusting the learning rate for each parameter based on the historical accumulation of gradients, AdaGrad can effectively allocate more learning resources to important features and navigate through valleys in the loss curve.

Dual averaging (AdaGrad) methods provide better generalization performance due to implicit regularization, which may positively affect early iterations while not negatively impacting the model's ability to fit the data during later "fine-tuning" epochs. The authors believe that further research into the effect of using stronger regularization at the early stages of optimization could be interesting more generally.

Momentum is a well-known and crucial component of deep learning optimization for various architectures and problem settings [46]. It is important to examine how momentum can be added to dual averaging updates and later to AdaGrad updates. The core idea of this algorithm is instead of evaluating the gradient at each step at the value of the argmin operation like in regular dual average, the gradient is evaluated at a moving average point, which smooths the iterate sequence. This is illustrated when momentum is added to SGD, where inline averaging is equivalent to more common equational forms of momentum for appropriate hyper-parameter choices.

Also the authors[7] propose a cube-root modification to maintain the right step size in their algorithm. This modification is inspired by a similar argument used for the standard square root method. Although the cube-root approach results in a final convergence rate bound that is not fully adaptive, the authors believe it's not a significant issue. The choice of step size still depends on other unknown factors, even when using a fully adaptive sequence.

In summary, the research introduces MADGRAD, a novel optimization technique for deep learning. MADGRAD builds upon the strengths of the AdaGrad optimizer and enhances its performance in various deep-learning tasks by combining adaptivity with excellent generalization capabilities. The experimental data presented by the authors shows that MADGRAD routinely outperforms state-of-the-art solutions for a wide range of realistic large-scale deep learning applications. They contend that the approach is a good starting point for optimizers spanning several machine learning subfields and a general-purpose optimizer for deep learning. The publication offers a new tool for researchers and practitioners to employ in their work, contributing to the expanding sub-field of optimization for deep learning.

2.9 BERT and RoBERTa

BERT, or Bidirectional Encoder Representations from Transformers [8], is primarily a pre-trained Transformer Encoder stack. To fully understand BERT, it is beneficial to refer to section 2.1 to understand encoders.

The Transformer model is a key innovation in natural language processing (NLP) that utilizes self-attention mechanisms to capture contextual relationships between words. BERT, being based on the Transformer architecture, leverages its power to learn contextual representations from large amounts of unlabeled text data.

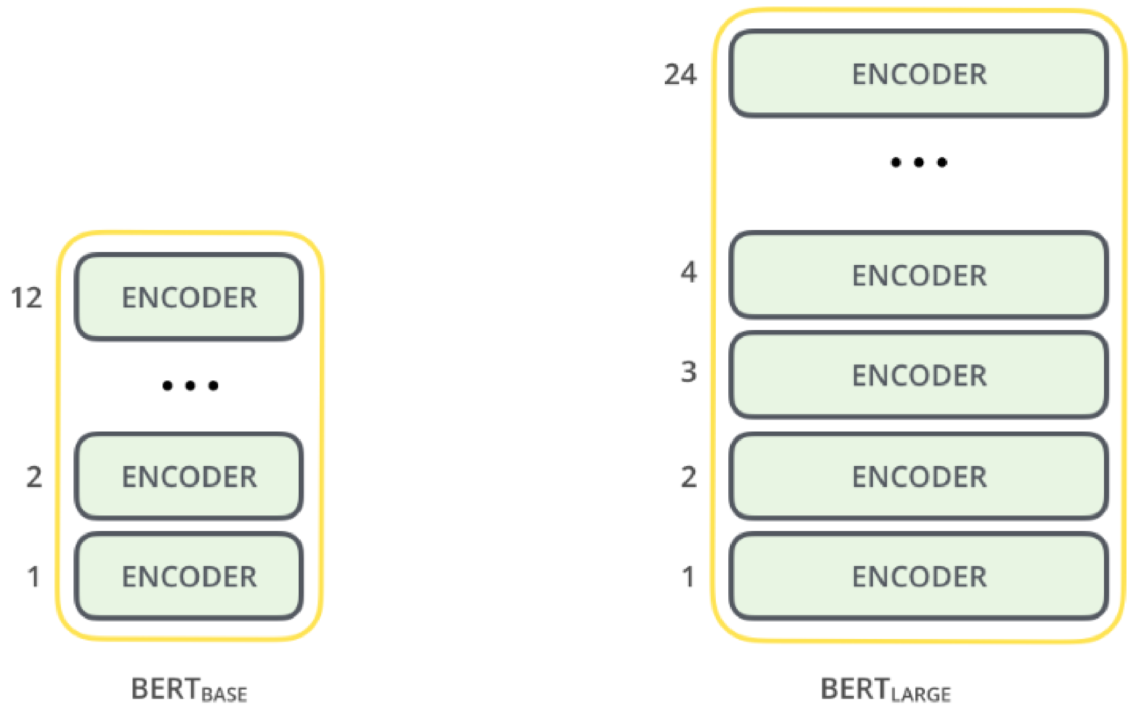


Fig. 2.9.1: BERT model

By pre-training on a vast corpus of text, BERT [8] learns to generate word embeddings that effectively capture the semantic meaning of words within their surrounding context. These pre-trained representations can then be fine-tuned on specific downstream tasks, such as question answering or text classification.

The BERT model comes in two sizes: Base and Large. Both versions have a significant number of encoder layers, also referred to as Transformer Blocks. The Base version consists of twelve encoder layers, while the Large version has twenty-four encoder layers.

Compared to the default configuration of the Transformer model described in the initial paper, BERT's encoder layers have larger feedforward networks. Specifically, the Base version has 768 hidden units in its feedforward network, while the Large version has 1024 hidden units.

Additionally, BERT [8] incorporates a greater number of attention heads compared to the default configuration. The Base version has 12 attention heads, while the

Large version utilizes 16 attention heads. In contrast, the original Transformer model described in the initial paper employed 6 encoder layers, 512 hidden units, and 8 attention heads as its default configuration.

The larger size of the BERT models, with more encoder layers, increased hidden units in the feedforward networks, and additional attention heads, allows for more complex and expressive representations to be learned from the text data. This increased capacity contributes to the enhanced performance of BERT on various NLP tasks, enabling it to capture more nuanced contextual relationships and achieve state-of-the-art results.

RoBERTa[24] is a widely embraced alternative and successor to BERT, offering significant improvements by optimizing the training hyperparameters for BERT in a meticulous and intelligent manner. Several straightforward modifications collectively enhance the performance of Roberta, surpassing BERT's capabilities across various tasks originally intended for BERT. Notably, at the time of Roberta's introduction, another influential transformer model called XLNet was also published. However, the changes introduced by XLNet are notably more complex and challenging to implement compared to the relatively straightforward modifications made in Roberta. This factor contributes to Roberta's popularity within the AI/NLP community.

Architecturally, Roberta follows the same structure as BERT. However, during the pretraining phase, Roberta exclusively employs Masked Language Modeling (MLM), whereas BERT additionally utilizes Next Sentence Prediction (NSP). Roberta introduces specific hyperparameter adjustments that contribute to its improved performance:

1. Longer training time and larger training data: Roberta benefits from an extended training duration and a substantial increase in training data, scaled up from 16 GB to 160 GB. 2. Larger batch size and vocabulary size: Roberta employs a larger batch size, increased from 256 to 8000, and a larger vocabulary size, expanded from 30,000 to 50,000 tokens. 3. Longer input sequences: Although Roberta employs longer input sequences, it still adheres to the maximum token limitation of 512 tokens, similar to BERT. 4. Dynamic masking: Roberta utilizes dynamic masking,

allowing for different masking patterns with each input sequence. In contrast, BERT employed the same masking pattern consistently.

CHAPTER 3

Related Works

In this section, we will delve into the investigation undertaken by DT-Fixup, as our thesis seeks to build upon and extend their established research.

3.1 Adam optimization

Adam (Adaptive Moment Estimation) [19] is an optimization algorithm that combines concepts from both adaptive learning rate methods and momentum-based methods. It has gained significant popularity in the field of deep learning due to its effectiveness in optimizing neural network models. Here's a review of the Adam optimizer:

1. Adaptive Learning Rates: Adam adapts the learning rate for each parameter individually. It uses estimates of the first and second moments of the gradients to dynamically adjust the learning rate during training. This adaptivity allows Adam to automatically adjust the learning rate based on the characteristics of each parameter, leading to efficient optimization.

2. Momentum: Adam incorporates momentum similar to other optimization algorithms such as SGD with Momentum. By including the momentum term, Adam accumulates past gradients and utilizes them to influence the current parameter update. This helps to smooth out the optimization process and accelerate convergence, especially in the presence of noisy or sparse gradients.

3. Convergence Speed: Adam is known for its fast convergence speed. It can quickly find good solutions, especially in deep learning tasks with large parameter spaces and complex loss landscapes. By adapting the learning rates and effectively

utilizing momentum, Adam efficiently navigates through the optimization landscape, making it a favored choice for many deep learning practitioners.

4. Robustness to Initial Learning Rate: Adam tends to be less sensitive to the choice of the initial learning rate compared to traditional optimization algorithms like SGD. This robustness alleviates the need for meticulous tuning of the learning rate, saving time and effort during model development.

5. Widely Used in Practice: Adam has become a popular choice for many researchers and practitioners in the deep learning community. It is widely supported by various deep learning frameworks, including TensorFlow and PyTorch, and often serves as a default or recommended optimizer for many tasks.

However, it is important to note that Adam may not always be the best optimizer for every scenario. In certain cases, such as in the presence of sparse gradients or for specific network architectures, other optimization algorithms may perform better.

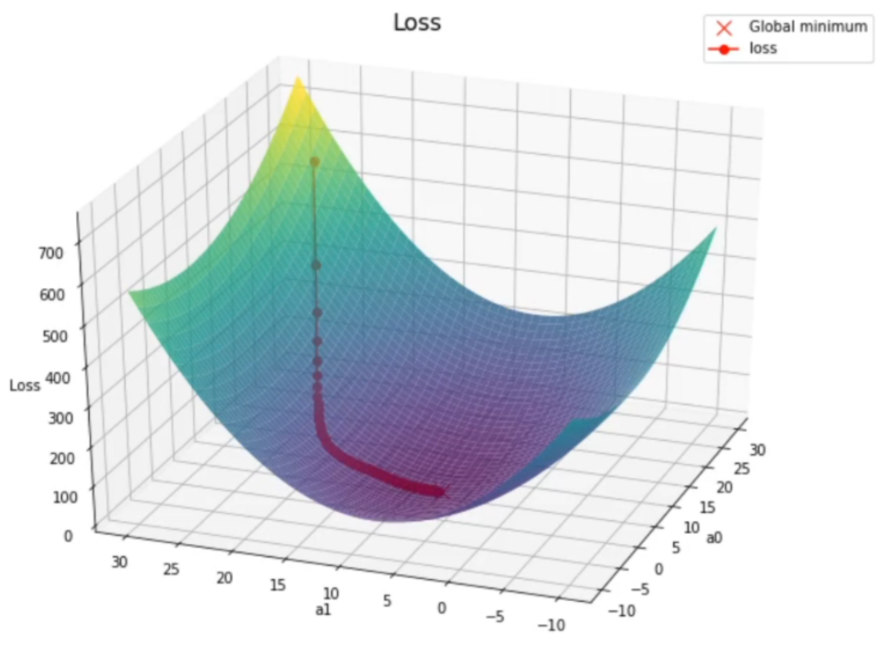


Fig. 3.1.1: ADAM Optimizer

In figure 3.1.1, the application of Adam optimization can be observed through its effect on the convergence of the loss curve. Adam is designed to iteratively update the model parameters in order to minimize the loss function.

In summary, Adam is a powerful and widely used optimization algorithm in deep learning. It combines adaptive learning rates with momentum to efficiently optimize neural network models, leading to fast convergence and robust performance.

3.2 DT-Fixup

The study explores the use of pre-trained language models in contemporary NLP systems for enhancing generalization when task-specific annotations are scarce, in particular large-scale models trained with transformers[54]. It is proposed that to enhance performance on tasks involving reasoning and structural comprehension, extra transformer layers should be used in conjunction with pre-trained models. Nonetheless, it is generally accepted that big datasets are necessary for training deep transformers from scratch, and very few attempts have been done using small datasets. This restricts the use of additional transformer layers over pre-trained models to increasingly difficult issues.

The Data-dependent Transformer Fixed-update initialization strategy (DT-Fixup) [54], which the authors suggest using, enables the training of substantially deeper transformers with greater generalization even on modest datasets. The foundation of DT-Fixup is a data-dependent initialization strategy that was developed by using various analyses to address numerous significant drawbacks of the T-Fixup method put out by Huang et al (2020) [14]. The authors [14] demonstrate that the optimization process, not the design, is to blame for the perception that deep transformers do not perform well on small datasets. It has been demonstrated that adding extra transformer layers during training can make it easier to learn complex relationships and data structures.

Spider [55], a challenging cross-domain Text-to-SQL semantic parsing benchmark, and ReClor [56], a reading comprehension dataset requiring logical thinking, is used to test the efficacy of DT-Fixup. Due to the low number of training samples in both datasets, large-batch training is undesirable due to its subpar generalization. With superior generalization and the ability to train substantially more complex transformer models, DT-Fixup demonstrates consistently outperforms the conventional method on both datasets.

By simply stacking transformer layers on top of RoBERTa, the authors of DT-Fixup achieve the second-place position for ReClor on the public leaderboard at the time of publication. Further error analysis reveals that improved generalization on the more challenging scenarios demanding reasoning and structural comprehension is mostly responsible for the performance improvements brought on by increasing the depth.

The overall architecture of the model implemented in their research is outlined as follows: they utilized the RoBERTa model and additional layers aiming to understand the effects of each model on the corresponding datasets. In figure 3.2.1, we show the overall architecture used in their paper. The inputs are initially fed into the RoBERTa large model for fine-tuning. Subsequently, they are processed through several transformer layers before being relayed to the output.

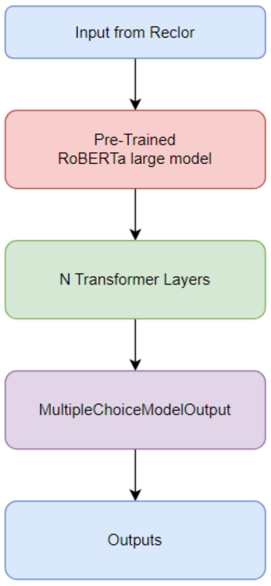


Fig. 3.2.1: Architecture of DT-Fixup

The DT-Fixup approach involves the following steps:

1. Utilize Xavier initialization[11] for all free parameters, excluding the weights loaded from pre-trained models.

2. Eliminate the learning rate warm-up and all layer normalization within the transformer layers, except for those in the pre-trained transformer.

3. Perform a forward pass on all training examples to obtain the maximum input norm μ, calculated as $\max x$, where x is the input.

4. Within each transformer layer, scale the MLP block by $(N-1)/2 * \mu$ for the vanilla transformer layer where N is the total number of layers in the model.

In step 4, we apply scaling since, when the input norm has the same magnitude as x, setting the encoder parameters to possess an equal norm and solving the corresponding equations leads to the same scale factors[54].

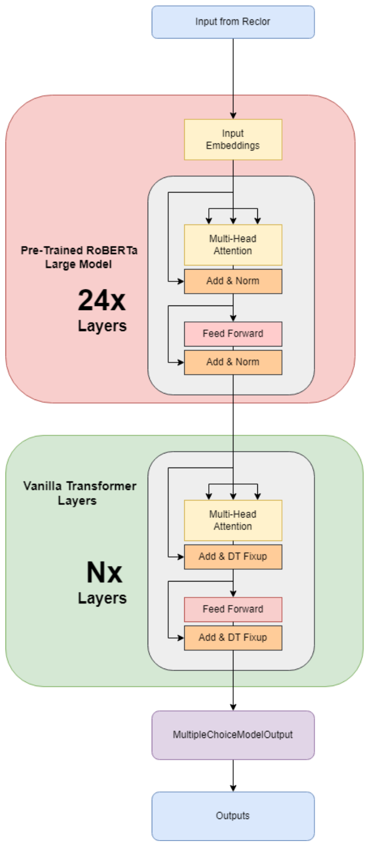


Fig. 3.2.2: Architecture of DT-Fixup (detailed)

In Figure 3.2.2, it's observable that the DT-Fixup is incorporated within the transformer block, represented by the green block. More specifically, the DT-Fixup is applied in the "Add and Normalize" segment, which is denoted in orange.

CHAPTER 4

Problem Statement and Methodology

4.1 Problem Statement

Since DT-Fixup's publication [54], has garnered over 30 citations, with 10 of them specifically focusing on applying the algorithm to Text-to-SQL datasets in various contexts. The remaining 20 citations primarily reference the paper to explain the suboptimal performance of transformers on smaller datasets. The significance of this lies in the fact that most specialized tasks have datasets that are hand-labelled and typically small in size, necessitating the use of logical strategies for effective performance[22]. Additionally, none of these researches comprehensively investigate the generalization capacity of the algorithm or scrutinize its impacts on different logical frameworks.

In this thesis, we expand the research on the DT-Fixup algorithm to different formal logical structures while also examining the impact of MADGRAD[7] optimization and power normalization[42] on the DT-Fixup algorithm. Through this comprehensive analysis, we hope to contribute valuable insights to the ongoing exploration of DT-Fixup and its potential applications in other domains.

Our study incorporates an additional dataset, LogiQA [22], for four main reasons. First, we aim to address the concern of generalization, as there's a lack of research exploring how our methodology adapts to diverse logical structures within similar datasets. This investigation will help determine the techniques' wider applicability on various logical structures, not just on the specific datasets already examined. Second, by evaluating other datasets, we can uncover potential shortcomings in current optimization methods if there are any, which might not be evident when assessed solely on one dataset. Such insights could guide future research and algorithmic enhancements. Third, the exploration of further datasets can reveal how these methods handle dataset-specific challenges and if they present any benefits over existing optimization strategies. Finally, incorporating more datasets into our analysis allows for a more thorough benchmarking of the algorithm, leading to a more holistic evaluation of its performance.

Specifically, we addressed the following research issues:

- We studied the performance of DT-Fixup on the MCQ reading comprehension dataset with different logical reasoning types.

  The datasets used in this thesis, namely ReClor[56] and LogiQA[22], evaluate natural language processing models' ability to understand and reason with language across a wide range of logical reasoning types.

  ReClor[56] tests abilities such as identifying necessary or sufficient assumptions, strengthening or weakening arguments, evaluating implications, and identifying flaws, among others. It also tests the ability to explain situations, identify principles, dispute issues, and understand the role of statements within larger arguments.

  The LogiQA[22] dataset, on the other hand, focuses on various types of logical reasoning like categorical reasoning, sufficient and necessary conditional reasoning, disjunctive reasoning, and conjunctive reasoning. These reasoning types involve categorizing concepts, analyzing conditional statements, handling disjunctive premises, and dealing with conjunctive premises.

  The investigation of these datasets allows the examination of the applicability of the DT-Fixup methodology to diverse datasets and reasoning types.

- One aspect we investigated was the role of the power normalization function in the performance of the DT-Fixup method.

Examining the effects of PowerNorm on small data architectures is vital for several reasons. Firstly, it is important to note that research has demonstrated that PN significantly surpasses LN in terms of performance for neural machine translation and language modelling[42]. However, no study has assessed its impact on smaller datasets (less than 10,000 samples) with complex logical structures, which is the primary focus of our research. PowerNorm is currently the only widely used normalization scheme in NLP, apart from LayerNorm, that is designed to surpass LayerNorm while employing a different normalization approach. Additionally, PowerNorm, as an adaptation of BatchNorm, maintains first-order smoothness, which is crucial when training datasets with limited data. Investigating PowerNorm's influence on small datasets can offer valuable insights into the effectiveness of this normalization method and its potential to improve the performance of models trained on limited data [36]. Also, in the context of DT-Fixup, since the normalization is responsible for finding the max of input norm $\|x\|$, which is ultimately used to compute the scaling factor used when the normalization is removed for training.

- The influence of the MADGRAD optimization algorithm on the effectiveness of the DT-Fixup method was investigated. We investigated the use of the MADGRAD optimization algorithm in the DTfixUp method, as proposed in [7]. MADGRAD is designed to improve the performance of deep learning models by addressing the limitations of existing optimization algorithms and has been shown to outperform other popular methods on several benchmark datasets and deep learning models. Meta AI's research on MADGRAD suggested a reasonable performance boost in both Computer Vision and Natural Language problems, which motivated us to test MADGRAD on the application of transformers on smaller datasets [7].

- The effectiveness of DT-Fixup, including its interactions with optimization algorithms, was investigated to determine the extent to which vanilla transformer layers impact its performance

We investigate MADGRAD (Momentumized, Adaptive, Dual averaged GRADient) method on DT-Fixup methodology because it is shown to be superior to Adam for several reasons according to the provided text:

1. Strong Generalization Performance: Adam, while widely used, has been critiqued for its potential to converge to poor local minima on certain critical problems, such as image classification. This has led to claims that adaptive methods like Adam don't generalize well. However, MADGRAD manages to combine adaptive behaviour with strong generalization, indicating it may avoid these pitfalls [7].
2. Consistency: MADGRAD consistently performs at a state-of-the-art level across various large-scale deep-learning problems like machine translation with a recurrent neural network and masked language modelling with a Transformer [7]. This consistency suggests a more robust performance profile than Adam, which might not be as consistently effective across different problem types. MADGRAD application on transformers is a novel idea that is not well researched.
3. No Additional Tuning Required: One major advantage of MADGRAD over Adam is that it doesn't necessitate more tuning than Adam.[7] This suggests that MADGRAD may be easier or at least as easy to use in practical applications, saving time and computational resources.
4. Evolution from AdaGrad: MADGRAD is a direct and systematic evolution from the dual averaging form of AdaGrad. This means that it incorporates the advantages of AdaGrad, a method that has a principled approach to diagonal adaptivity, while also incorporating improvements suited for deep learning optimization [7].
5. Momentum and Stabilization: While Adam also incorporates momentum and bias correction, the text implies that MADGRAD might have further optimized these aspects for better performance compared to Adam [7].

Understanding the advantages and disadvantages of DT-Fixup will be made easier with the help of our study, which will also guide future studies along this approach.

4.2 Methodology

In this dissertation, we've constructed the DT-Fixup architecture from scratch which is not available in their (Dt-Fixup) original research paper.

In our research, we are not introducing a novel methodology per se. Instead, we are building upon the existing DT-Fixup framework. Our approach consists of experimentation with the modification of certain variables(optimization and normalization) within this established algorithm to investigate and understand its potential impacts and improvements.

Our work primarily relies on two models: the RoBERTa base, which we used to fine-tune the LogiQA dataset, and the RoBERTa large, which we used to fine-tune the ReClor dataset. Our goal was to examine the influence of both RoBERTa models on their respective datasets.

The pipeline for our research, which includes a detailed implementation of the DT-Fixup algorithm, is comprehensively presented in our code. We developed a custom class that not only integrates the DT-Fixup architecture but also incorporates PowerNorm and MADGRAD, demonstrating our extension and exploration of these techniques within the DT-Fixup algorithm.

CHAPTER 5

Experiments and Results

5.1 Setup

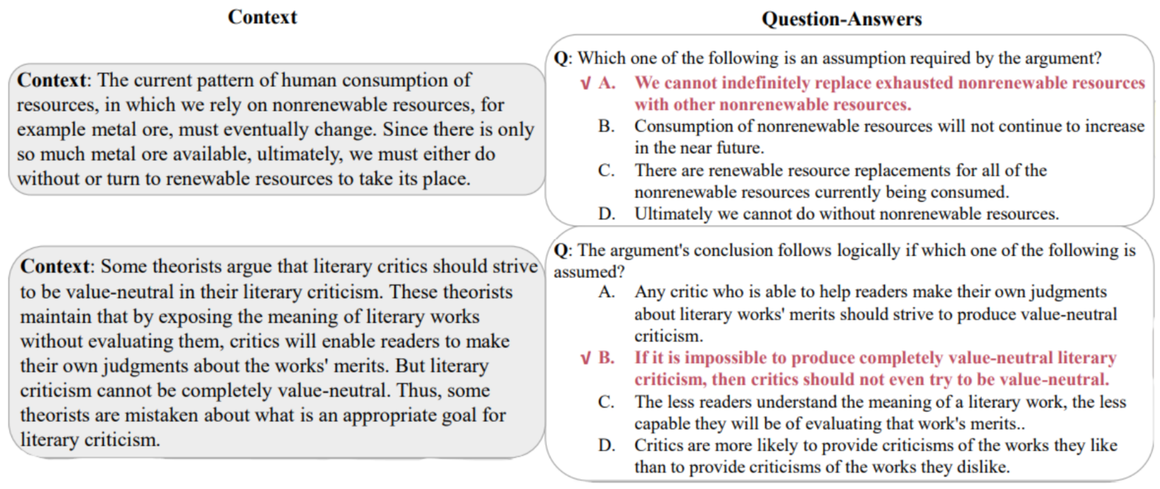
5.1.1 Datasets

We utilized two of the datasets accessible for multiple-choice reading comprehension in the context of testing logic, namely LogiQA[22] and ReClor[56]. As previously mentioned in the background section, these datasets contain various logical structures. This diversity in data will aid us in exploring how well the DT-Fixup algorithm can generalization.

We use the RoBERTa large model for pretraining on the ReClor dataset and the RoBERTa base model for pretraining on the LogiQA dataset.

5.1.2 ReClor: A Reading Comprehension Dataset Requiring Logical Reasoning

ReClor [56] is a dataset created to evaluate models' capacity for logical inference in multiple-choice reading comprehension tests. ReClor consists of 6,138 data points, each of which has a context, a question, and four possible answers, only one of which is the best one. The data points were chosen from standardized tests like the GMAT and LSAT, which call for intricate logical reasoning, and the information was gathered from public websites and literature. Figure 2.2.1 shows some sample data points from the dataset.

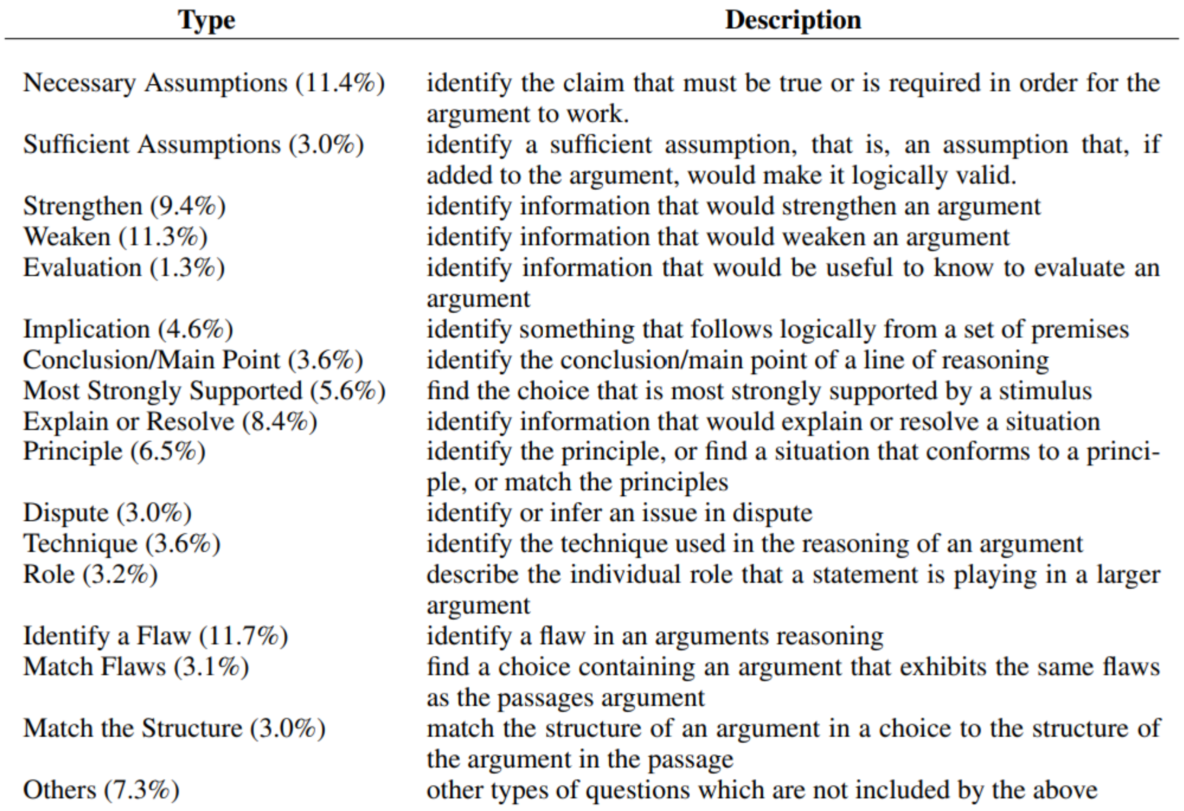


Context	Question-Answers
Context: The current pattern of human consumption of resources, in which we rely on nonrenewable resources, for example metal ore, must eventually change. Since there is only so much metal ore available, ultimately, we must either do without or turn to renewable resources to take its place.	**Q**: Which one of the following is an assumption required by the argument? √ A. **We cannot indefinitely replace exhausted nonrenewable resources with other nonrenewable resources.** B. Consumption of nonrenewable resources will not continue to increase in the near future. C. There are renewable resource replacements for all of the nonrenewable resources currently being consumed. D. Ultimately we cannot do without nonrenewable resources.
Context: Some theorists argue that literary critics should strive to be value-neutral in their literary criticism. These theorists maintain that by exposing the meaning of literary works without evaluating them, critics will enable readers to make their own judgments about the works' merits. But literary criticism cannot be completely value-neutral. Thus, some theorists are mistaken about what is an appropriate goal for literary criticism.	**Q**: The argument's conclusion follows logically if which one of the following is assumed? A. Any critic who is able to help readers make their own judgments about literary works' merits should strive to produce value-neutral criticism. √ B. **If it is impossible to produce completely value-neutral literary criticism, then critics should not even try to be value-neutral.** C. The less readers understand the meaning of a literary work, the less capable they will be of evaluating that work's merits.. D. Critics are more likely to provide criticisms of the works they like than to provide criticisms of the works they dislike.

Fig. 5.1.1: Examples of some question types from ReClor. The correct options are marked in red[56]

Using 4,638, 500, and 1,000 data points each, the data are divided into a training set, a validation set, and a testing set.

There are 17 categories of logical thinking questions in the ReClor dataset which is shown in figure 2.2.2[56]. The most common ones are Necessary Assumptions, Sufficient Assumptions, Weaken, Identify a Flaw, etc.



Type	Description
Necessary Assumptions (11.4%)	identify the claim that must be true or is required in order for the argument to work.
Sufficient Assumptions (3.0%)	identify a sufficient assumption, that is, an assumption that, if added to the argument, would make it logically valid.
Strengthen (9.4%)	identify information that would strengthen an argument
Weaken (11.3%)	identify information that would weaken an argument
Evaluation (1.3%)	identify information that would be useful to know to evaluate an argument
Implication (4.6%)	identify something that follows logically from a set of premises
Conclusion/Main Point (3.6%)	identify the conclusion/main point of a line of reasoning
Most Strongly Supported (5.6%)	find the choice that is most strongly supported by a stimulus
Explain or Resolve (8.4%)	identify information that would explain or resolve a situation
Principle (6.5%)	identify the principle, or find a situation that conforms to a principle, or match the principles
Dispute (3.0%)	identify or infer an issue in dispute
Technique (3.6%)	identify the technique used in the reasoning of an argument
Role (3.2%)	describe the individual role that a statement is playing in a larger argument
Identify a Flaw (11.7%)	identify a flaw in an arguments reasoning
Match Flaws (3.1%)	find a choice containing an argument that exhibits the same flaws as the passages argument
Match the Structure (3.0%)	match the structure of an argument in a choice to the structure of the argument in the passage
Others (7.3%)	other types of questions which are not included by the above

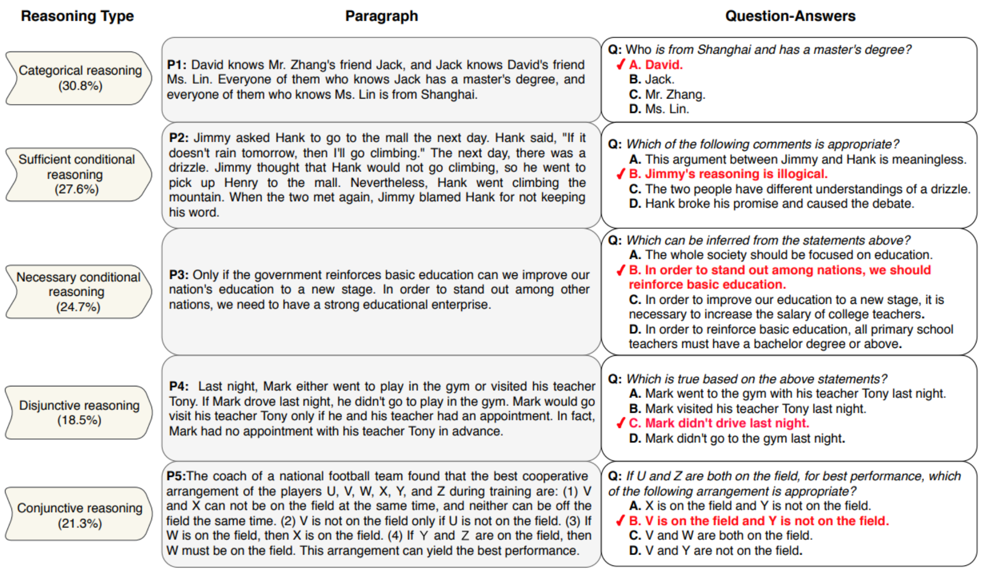
Fig. 5.1.2: These percentages and descriptions represent various types of logical reasoning. [56]

We talk about this here because the logical reasoning required to solve ReClor is very different from the logical reasoning used in LogiQA.

5.1.3 LogiQA

LogiQA [22] comprises 8,678 paragraph-question pairs, each with four potential answers. The dataset is derived from publicly accessible logical examination papers meant for reading comprehension. These papers are crafted by domain experts with the aim of assessing the logical reasoning capabilities of test takers. As such, the reliability and topical scope of the questions is commendable. Problems from the original dataset were meticulously selected, excluding those that incorporate figures, and charts or are excessively mathematical, thereby ensuring a broad representation of logical reasoning types.

The LogiQA paper outlines five distinct types of logical reasoning (shown in figure 2.2.3 with examples) used in the dataset [22] like Categorical reasoning, Sufficient conditional reasoning, Necessary conditional reasoning, Disjunctive reasoning, and Conjunctive reasoning.



Reasoning Type	Paragraph	Question-Answers
Categorical reasoning (30.8%)	**P1:** David knows Mr. Zhang's friend Jack, and Jack knows David's friend Ms. Lin. Everyone of them who knows Jack has a master's degree, and everyone of them who knows Ms. Lin is from Shanghai.	**Q:** Who *is from Shanghai and has a master's degree?* ✓ **A. David.** **B.** Jack. **C.** Mr. Zhang. **D.** Ms. Lin.
Sufficient conditional reasoning (27.6%)	**P2:** Jimmy asked Hank to go to the mall the next day. Hank said, "If it doesn't rain tomorrow, then I'll go climbing." The next day, there was a drizzle. Jimmy thought that Hank would not go climbing, so he went to pick up Henry to the mall. Nevertheless, Hank went climbing the mountain. When the two met again, Jimmy blamed Hank for not keeping his word.	**Q:** *Which of the following comments is appropriate?* **A.** This argument between Jimmy and Hank is meaningless. ✓ **B. Jimmy's reasoning is illogical.** **C.** The two people have different understandings of a drizzle. **D.** Hank broke his promise and caused the debate.
Necessary conditional reasoning (24.7%)	**P3:** Only if the government reinforces basic education can we improve our nation's education to a new stage. In order to stand out among other nations, we need to have a strong educational enterprise.	**Q:** *Which can be inferred from the statements above?* **A.** The whole society should be focused on education. ✓ **B. In order to stand out among nations, we should reinforce basic education.** **C.** In order to improve our education to a new stage, it is necessary to increase the salary of college teachers. **D.** In order to reinforce basic education, all primary school teachers must have a bachelor degree or above.
Disjunctive reasoning (18.5%)	**P4:** Last night, Mark either went to play in the gym or visited his teacher Tony. If Mark drove last night, he didn't go to play in the gym. Mark would go visit his teacher Tony only if he and his teacher had an appointment. In fact, Mark had no appointment with his teacher Tony in advance.	**Q:** *Which is true based on the above statements?* **A.** Mark went to the gym with his teacher Tony last night. **B.** Mark visited his teacher Tony last night. ✓ **C. Mark didn't drive last night.** **D.** Mark didn't go to the gym last night.
Conjunctive reasoning (21.3%)	**P5:**The coach of a national football team found that the best cooperative arrangement of the players U, V, W, X, Y, and Z during training are: (1) V and X can not be on the field at the same time, and neither can be off the field the same time. (2) V is not on the field only if U is not on the field. (3) If W is on the field, then X is on the field. (4) If Y and Z are on the field, then W must be on the field. This arrangement can yield the best performance.	**Q:** *If U and Z are both on the field, for best performance, which of the following arrangement is appropriate?* **A.** X is on the field and Y is not on the field. ✓ **B. V is on the field and Y is not on the field.** **C.** V and W are both on the field. **D.** V and Y are not on the field.

Fig. 5.1.3: Examples of LogiQA's logical reasoning types. Red ticks indicate the correct answer [22]

5.1.4 Evaluation Strategy and Metrics

When it comes to evaluating how well a model performs multiple metrics can be used. In the present thesis work, the common metric, accuracy, is used. An evaluation statistic called accuracy is defined by the percentage of the number of samples that are correctly predicted by the model.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{1}$$

In classification modelling, it's crucial to assess the model's accuracy in predicting the right result.

5.2 Results

In this section, we explain our findings in response to our research questions.

5.2.1 Baseline Experiments

We will analyze the experiments carried out using solely transformer layers on both the ReClor and LogiQA datasets and gain a comprehensive understanding of the outcomes. In addition, we will also explore the DT-Fixup technique and its impact on the experiments with transformer layers on both the ReClor and LogiQA datasets.

Baseline Experiment		
Dataset	Our Result	Published Result [56] [22]
ReClor	60.2	62.6
LogiQ	32.25	35.85

Although we utilized a very similar model to that published in the papers, there can be several reasons why our results are lower than the published results for the same experiment. These reasons include:

- Difference in experimental setup: The experiment setup used by you and the published results may differ in the form of the training data used, evaluation metric, feature extraction, etc. Even minor differences in the experimental setup can cause significant differences in the results obtained.
- Difference in implementation: Even if the experimental setup is identical, the implementation of the model and the methodology used to obtain the results can vary. This can be in the form of hyperparameter settings, optimization algorithms, regularization techniques, and so on.

- Randomness and Variability: The difference in results could be due to the randomness and variability in the experiment. For instance, if the experiment involves training a deep learning model, the model's performance can vary significantly based on the initialization of the model's parameters.

- Number of epochs and training time: The number of epochs used for training, as well as the training time, can significantly impact the performance of the model.

- Hardware differences: The GPU or other hardware used for training the model may be different in terms of its capabilities, and this can impact the performance of the model.

- Repetition of experiments: If the published results were obtained by running the experiment multiple times, this could reduce the effect of random variations and help to obtain more stable and reliable results.

5.2.1.1 ReClor Baseline Experiment

We conducted several experiments to evaluate the impact of our research on transformers. To establish a benchmark for the performance of the original transformer model on the ReClor datasets, we ran several baseline tests without making any modifications to the model. These baseline tests allowed us to compare the performance of the modified transformer models with that of the original model and to identify any potential improvements or limitations resulting from the modifications. From the results in the table for the ReClor dataset below, it can be seen that adding one additional transformer layer to the RoBERTa large model improves the performance significantly, with a validation accuracy of 60.646% and a test accuracy of 64.6%. However, adding more layers does not lead to further improvement, and in fact, the performance drops dramatically with three layers or more.

Baseline Experiment for ReClor			
Model Name	Transformer Layers	Validation Accuracy	Test Accuracy
RoBERTa large	0	60.2	43.6
RoBERTa large	1	60.646	64.6
RoBERTa large	2	63	63
RoBERTa large	3	21.8	21.8
RoBERTa large	4	59.4	59.4

The accuracy curve for the ReClor Baseline Experiment is shown in Figure 5.2.1, where the x-axis represents the number of epochs, and the y-axis represents the accuracy. The results indicate that adding transformer layers to the model tends to improve the accuracy of the model. However, it was observed that if too many transformer layers are added, the model finds it difficult to decrease the loss or fine-tune the parameters effectively. These findings suggest that adding transformer layers to the ReClor model can improve its accuracy, but a careful balance must be struck to avoid overfitting and ensure efficient training of the model.

The best performance is achieved with the RoBERTa large model with one additional transformer layer, which outperforms the original RoBERTa large model by a significant margin. These findings suggest that the addition of one transformer layer can be an effective modification for improving the performance of the RoBERTa large model on the ReClor dataset.

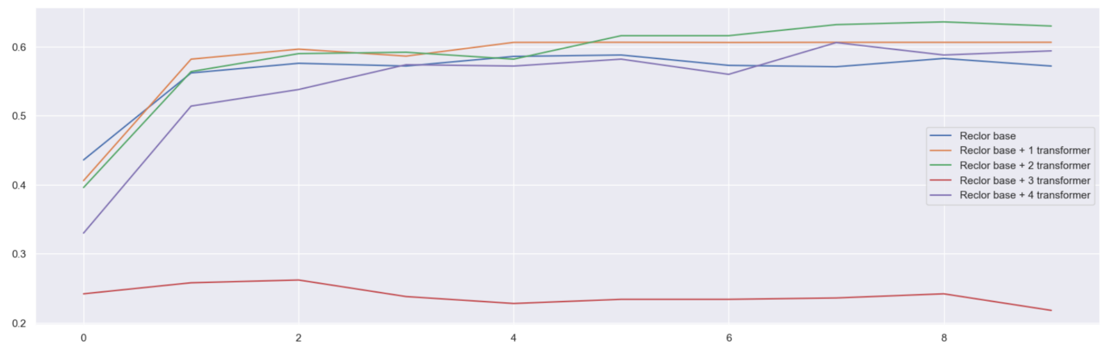


Fig. 5.2.1: Accuracy curve for ReClor Baseline Experiment

However, adding more layers does not lead to further improvement, and in fact, the performance drops dramatically with three layers or more. The best performance is achieved with the RoBERTa large model with one additional transformer layer, which outperforms the original RoBERTa large model by a significant margin. These findings suggest that the addition of one transformer layer can be an effective modification for improving the performance of the RoBERTa large model on the ReClor dataset.

5.2.1.2 LogiQA Baseline Experiment

From the results in the table for the LogiQA dataset, it can be observed that adding transformer layers to the RoBERTa base model has a relatively small impact on the performance which can be noticed from the table below. The addition of one transformer layer results in a slight improvement in the validation accuracy, but the test accuracy remains largely the same. Adding two transformer layers further improves the performance, with a validation accuracy of 34.715% and a test accuracy of 34.715%.

Baseline Experiment for LogiQA			
Model Name	Transformer Layer	Validation Accuracy	Test Accuracy
RoBERTa base	0	32.25	32.25
RoBERTa base	1	33.6	32.71
RoBERTa base	2	34.715	34.715
RoBERTa base	3	34.01	34.25
RoBERTa base	4	34.40	34.4086

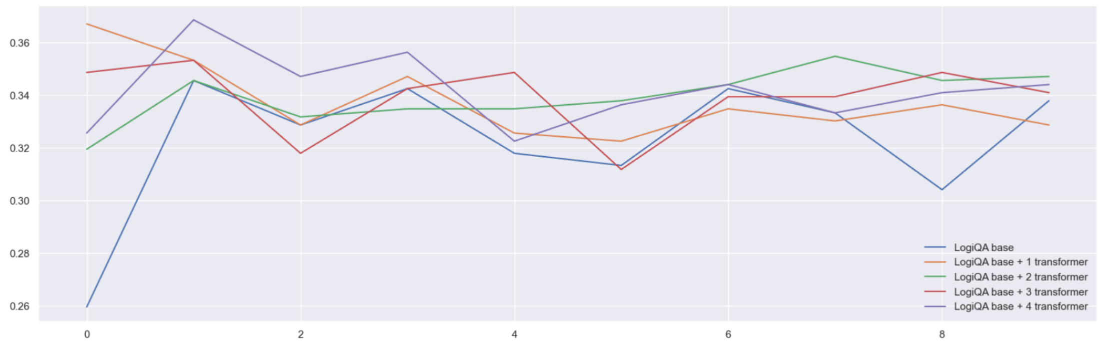


Fig. 5.2.2: Accuracy curve for LogiQA Baseline Experiment

The accuracy curve for the LogiQA Baseline Experiment is depicted in Fig. 5.2.2, where the x-axis represents the number of epochs and the y-axis represents the accuracy. It can be observed that adding transformer layers to the RoBERTa base model leads to a slight increase in accuracy. However, the performance gain is not substantial, and adding more transformer layers does not lead to significant improvements. In fact, the performance of the model fluctuates with three or four transformer layers.

The results suggest that the optimal number of transformer layers to add to the RoBERTa base model for the LogiQA dataset is two. The model with two additional transformer layers performs slightly better than the original RoBERTa base model and yields the best performance overall. These findings highlight the importance of carefully selecting the number of transformer layers to add to a base model to achieve optimal performance on a given dataset.

5.2.1.3 ReClor with DT-Fixup Baseline Experiment

In this experiment, we evaluated the performance of the ReClor model with DT-Fixup using RoBERTa large and varying numbers of transformer layers. The validation and test accuracy were used as the evaluation metrics for the models. The RoBERTa large model achieved a validation accuracy of 60.2% and a test accuracy of 43.6%.

Baseline Experiment for ReClor with DT-Fixup

Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa large	0	None	60.2	43.6
RoBERTa large	1	DT-Fixup	61.00	61.00
RoBERTa large	2	DT-Fixup	59.00	59.00
RoBERTa large	3	DT-Fixup	58.6	58.6
RoBERTa large	4	DT-Fixup	59	59

By adding 1 transformer layer and DT-Fixup to the RoBERTa large model, we achieved an improvement in validation accuracy to 61.00%, which was also maintained in the test accuracy. With 2 transformer layers and DT-Fixup, we achieved a validation accuracy of 59.00% and a test accuracy of 59.00%. However, the addition of 3 transformer layers and DT-Fixup slightly decreased the validation accuracy to 58.6% and the test accuracy remained the same. Finally, by adding 4 transformer layers and DT-Fixup, we achieved a validation accuracy of 59% and a test accuracy of 59%.

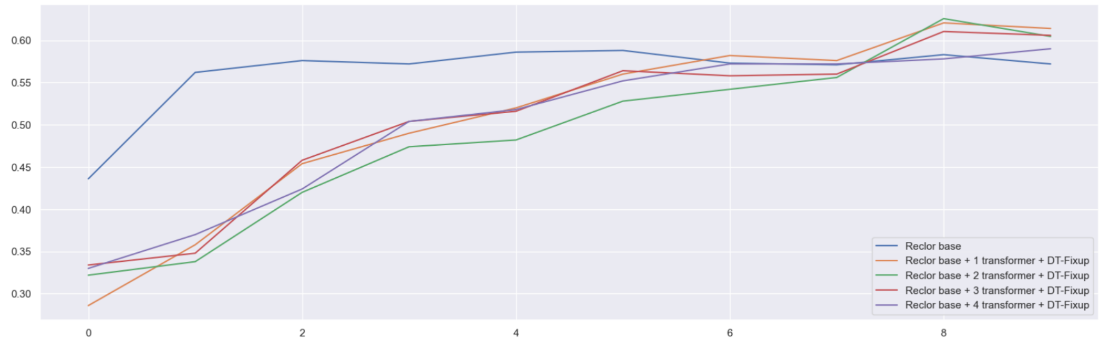


Fig. 5.2.3: Accuracy curve for Baseline Experiment for ReClor with DT-Fixup

The graph shown in Fig. 5.2.3 represents the accuracy curve for the Baseline Experiment conducted on the ReClor model with DT-Fixup. The X-axis indicates the number of epochs, while the Y-axis represents the accuracy achieved. The graph indicates that the inclusion of transformer layer with DT-Fixup slightly improved the accuracy. However, the addition of more layers did not lead to a significant improvement in performance, and the accuracy fluctuated with three or four layers. Therefore, it can be concluded that the ReClor model with DT-Fixup performed better with the inclusion of one or two transformer layers, and increasing the number of transformer layers did not result in any significant improvement and may even slightly decrease the accuracy.

5.2.1.4 LogiQA with DT-Fixup Baseline Experiment

As we analyzed the baseline experiment for LogiQA with DT-Fixup, we observed that the RoBERTa base model achieved a validation accuracy and test accuracy of 32.25.

Baseline Experiment for LogiQA with DT-Fixup				
Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa base	0	None	32.25	32.25
RoBERTa base	1	DT-Fixup	40.8	31.33
RoBERTa base	2	DT-Fixup	34.5	31.79
RoBERTa base	3	DT-Fixup	33.83	31.18
RoBERTa base	4	DT-Fixup	34.16	31.49

However, when we added one transformer layer and DT-Fixup, the validation accuracy increased to 40.8, while the test accuracy decreased to 31.33. Similarly, when we added two transformer layers and DT-Fixup, the validation accuracy decreased to 34.5, and the test accuracy decreased to 31.79. The same pattern was observed for models with three and four transformer layers and DT-Fixup, with validation accuracies of 33.83 and 34.16, and test accuracies of 31.18 and 31.49, respectively. These results suggest that adding DT-Fixup to the RoBERTa base model and transformer layers may not always lead to better performance in LogiQA.

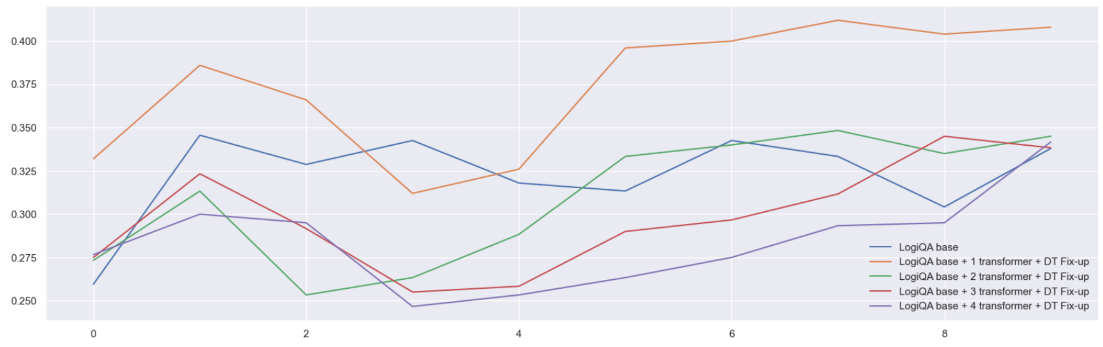


Fig. 5.2.4: Accuracy curve for Baseline Experiment for LogiQA with DT-Fixup

The accuracy curve for the Baseline Experiment conducted on the LogiQA model with DT-Fixup is shown in Fig. 5.2.4. The X-axis represents the number of epochs, and the Y-axis shows the accuracy achieved. It can be observed that adding a transformer layer with DT-Fixup slightly increased the accuracy. However, the inclusion of more layers did not lead to a significant improvement in performance, and the accuracy fluctuated with three or four layers. Therefore, it can be concluded that the LogiQA model with DT-Fixup performed better with the addition of one or two transformer layers, and increasing the number of transformer layers did not result in any significant improvement and may even slightly decrease the accuracy.

5.2.2 Normalization's Impact on DT-Fixup Performance

We conducted a thorough analysis of the experimental results obtained through the use of transformer layers on both the ReClor and LogiQA datasets. Furthermore, we investigated the effects of DT-Fixup and PowerNorm techniques on these experiments. By doing so, we aimed to gain a more comprehensive understanding of the performance of the transformer layers in these contexts.

5.2.2.1 ReClor with PowerNorm Baseline Experiment

In this experiment, we evaluated the effectiveness of ReClor with PowerNorm on RoBERTa large models with varying numbers of transformer layers. The goal was to determine if this combination could improve the accuracy of the models on a validation and test dataset.

We presented the results in a table, which includes the model name, validation accuracy, and test accuracy for each variation of the RoBERTa large model with PowerNorm and a different number of transformer layers.

Baseline Experiment for ReClor with PowerNorm				
Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa large	0	None	60.2	43.6
RoBERTa large	1	PowerNorm	46.33	55.6
RoBERTa large	2	PowerNorm	31.0	37.2
RoBERTa large	3	PowerNorm	32.33	38.8
RoBERTa large	4	PowerNorm	26.0	31.2

The results show that the baseline RoBERTa large model achieved the highest validation accuracy at 60.2%, but a lower test accuracy at 43.6%. The RoBERTa large model with one transformer layer and PowerNorm achieved the highest test accuracy at 55.6%, but a lower validation accuracy at 46.33%. The models with two and three transformer layers did not showed improvements in accuracy with the addition of PowerNorm, also the model with four transformer layers saw a decline in performance.

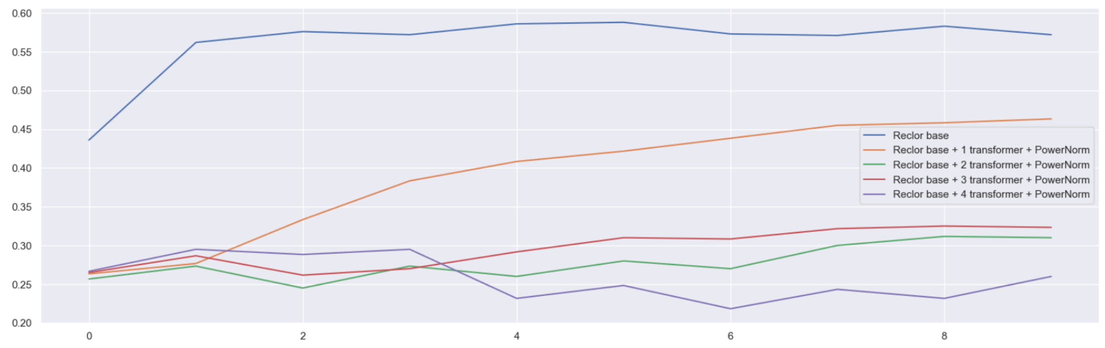


Fig. 5.2.5: Baseline Experiment for ReClor with PowerNorm

The results of the baseline experiment shown in Fig. 5.2.5 suggest that the use of ReClor with PowerNorm does not lead to improved accuracy in RoBERTa large models with a moderate number of transformer layers. To determine the best configuration for different types of datasets and tasks, additional experiments can be conducted to explore the ideal number of transformer layers and PowerNorm settings on other datasets.

5.2.2.2 LogiQA with PowerNorm Baseline Experiment

We conducted an experiment to evaluate the effectiveness of LogiQA with PowerNorm on RoBERTa base models with varying numbers of transformer layers. The goal was to determine if this combination could improve the accuracy of the models on a validation and test dataset.

The results of the experiment are presented in a table, which includes the model name, validation accuracy, and test accuracy for each variation of the RoBERTa base model with PowerNorm and a different number of transformer layers.

Baseline Experiment for LogiQA with PowerNorm				
Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa base	0	None	32.25	32.25
RoBERTa base	1	PowerNorm	42.2	32.41
RoBERTa base	2	PowerNorm	35.8	26.57
RoBERTa base	3	PowerNorm	39.4	30.10
RoBERTa base	4	PowerNorm	35.8	24.3

From the table, it is clear that the RoBERTa base model with one transformer layer and PowerNorm achieved the highest validation accuracy of 42.2%. However, the test accuracy for this model was only marginally higher than the baseline at 32.41%. The models with two and three transformer layers also showed some improvement in validation accuracy but saw a decline in test accuracy. The model with four transformer layers saw a significant decline in both validation and test accuracy.

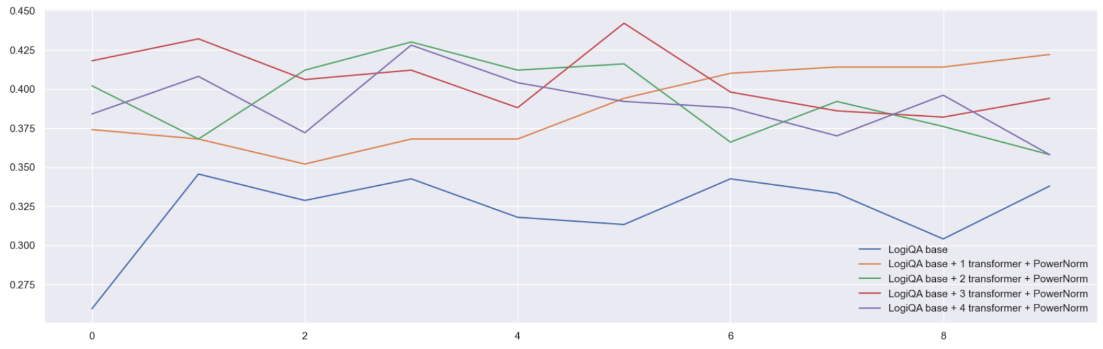


Fig. 5.2.6: Baseline Experiment for LogiQA with PowerNorm

In Fig. 5.2.6, we can see the results of a baseline experiment for LogiQA with PowerNorm. The X-axis represents the number of epochs, while the Y-axis shows the accuracy. The experiment suggests that using LogiQA with PowerNorm may be effective in improving the accuracy of RoBERTa base models for this specific task. However, the addition of PowerNorm and more transformer layers did not consistently improve performance on the validation and test datasets. If you plan to conduct further experiments, it may be worth exploring different combinations of pre-processing techniques and models for LogiQA tasks.

5.2.2.3 ReClor with DT-Fixup and PowerNorm Baseline Experiment

We experimented to evaluate the effectiveness of using DT-Fixup and PowerNorm in combination with RoBERTa large models with varying numbers of transformer layers for the ReClor task. The objective was to determine if this combination could improve the accuracy of the models on a validation and test dataset.

The results of the experiment are presented in a table, which includes the model name, validation accuracy, and test accuracy for each variation of the RoBERTa large model with DT-Fixup and PowerNorm and a different number of transformer layers.

Baseline Experiment for ReClor with DT-Fixup and PowerNorm				
Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa large	0	None	60.2	43.6
RoBERTa large	1	DT-Fixup + PowerNorm	50.83	61.00
RoBERTa large	2	DT-Fixup + PowerNorm	48.33	58.00
RoBERTa large	3	DT-Fixup + PowerNorm	48.83	58.6
RoBERTa large	4	DT-Fixup + PowerNorm	50.83	61.00

From the table, it is evident that the addition of DT-Fixup and PowerNorm to the RoBERTa large model improved the accuracy of the models on both validation and test datasets. The model with one transformer layer achieved the highest test accuracy of 61.00% with a validation accuracy of 50.83%. The models with two and three transformer layers also showed improvements in test accuracy, but at the cost of lower validation accuracy. The model with four transformer layers had the same accuracy as the model with one transformer layer

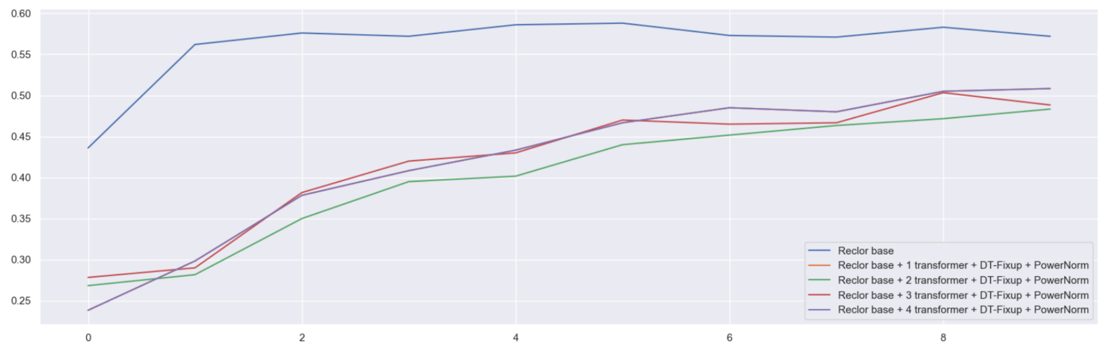


Fig. 5.2.7: Baseline Experiment for ReClor with DT-Fixup and PowerNorm

Fig. 5.2.7 shows the results of a baseline experiment for ReClor with DT-Fixup and PowerNorm, where the X-axis represents the number of epochs, and the Y-axis represents accuracy. The experiment suggests that combining DT-Fixup and PowerNorm may be effective in improving the accuracy of RoBERTa large models for the ReClor task. Moreover, adding DT-Fixup and PowerNorm consistently improved test accuracy across models with different numbers of transformer layers compared to just PowerNorm layers added. Further experiments could explore the effectiveness of combining DT-Fixup and PowerNorm with other pre-processing techniques and models.

5.2.2.4 LogiQA with DT-Fixup and PowerNorm Baseline Experiment

In this study, we conducted a baseline experiment for LogiQA with DT-Fixup and PowerNorm, using different variations of the RoBERTa base model with additional transformer layers. The purpose of this experiment was to evaluate the impact of DT-Fixup and PowerNorm on the model's performance in the LogiQA task.

Baseline Experiment for LogiQA with DT-Fixup and PowerNorm				
Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa base	0	None	60.2	32.25
RoBERTa base	1	DT-Fixup + PowerNorm	34.0	31.33
RoBERTa base	2	DT-Fixup + PowerNorm	34.5	31.79
RoBERTa base	3	DT-Fixup + PowerNorm	33.833	31.182
RoBERTa base	4	DT-Fixup + PowerNorm	34.166	31.49

The results of the experiment are summarized in the table above. The RoBERTa base model achieved a validation accuracy and test accuracy of 32.25%. The models with additional transformer layers and DT-Fixup and PowerNorm showed slight improvements in validation accuracy, but their test accuracy was lower than the base model. The best-performing model was the RoBERTa base model with 2 transformer layers and DT-Fixup and PowerNorm, achieving a validation accuracy of 34.5% and a test accuracy of 31.79%.

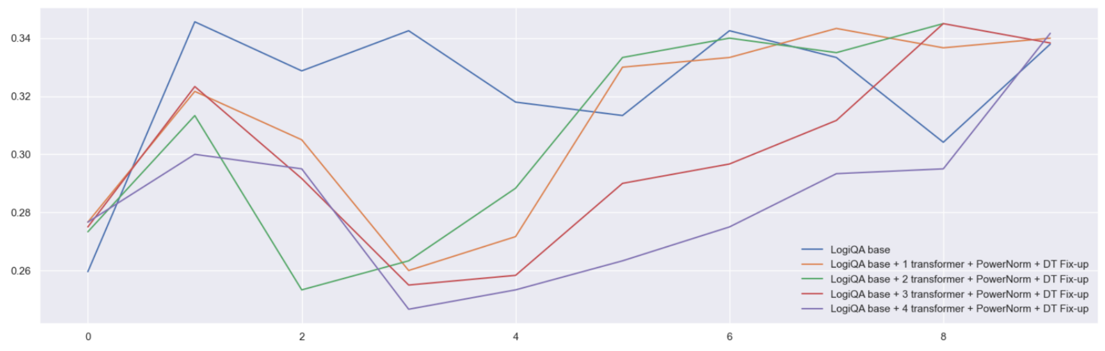


Fig. 5.2.8: Baseline Experiment for LogiQA with DT-Fixup and PowerNorm

In Fig. 5.2.7, we present the results of a baseline experiment for ReClor with DT-Fixup and PowerNorm. The X-axis represents the number of epochs, while the Y-axis shows the accuracy. Our findings suggest that adding DT-Fixup and PowerNorm to the RoBERTa base model does not consistently improve its performance in the LogiQA task. We believe that further experimentation and analysis are necessary to determine the optimal configuration of the model for this task.

5.3 Optimization Algorithms and DT-Fixup Methodology

We will perform an in-depth analysis of the experimental outcomes achieved through the implementation of transformer layers on the ReClor and LogiQA datasets. Additionally, we will examine the impacts of both DT-Fixup and MADGRAD techniques on these experiments. Our objective is to attain a thorough comprehension of the performance of transformer layers in these particular settings.

5.3.0.1 ReClor with MADGRAD Baseline Experiment

In this experiment, we evaluated the performance of RoBERTa large model and its variations with MADGRAD optimizer for the task of ReClor. The results are presented in the table above.

The baseline model, RoBERTa large, achieved a high validation accuracy of

60.2% and a test accuracy of 43.6%. However, upon adding transformer layers with MADGRAD optimizer, the performance of the model degraded significantly. The RoBERTa large model with one additional transformer layer and MADGRAD optimizer achieved a validation accuracy of 23.16% and a test accuracy of 27.8%. The models with two, three, and four additional transformer layers also experienced a decrease in performance, achieving validation accuracies of 19.66%, 20.5%, and 18.0%, respectively, with corresponding test accuracies of 23.6%, 24.6%, and 21.6%.

Baseline Experiment for ReClor with MADGRAD				
Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa large	0	None	60.2	43.6
RoBERTa large	1	MADGRAD	23.16	27.8
RoBERTa large	2	MADGRAD	19.66	23.6
RoBERTa large	3	MADGRAD	20.5	24.6
RoBERTa large	4	MADGRAD	18.0	21.6

However, upon adding transformer layers with MADGRAD optimizer, the performance of the model degraded significantly. The RoBERTa large model with one additional transformer layer and MADGRAD optimizer achieved a validation accuracy of 23.16% and a test accuracy of 27.8%. The models with two, three, and four additional transformer layers also experienced a decrease in performance, achieving validation accuracies of 19.66%, 20.5%, and 18.0%, respectively, with corresponding test accuracies of 23.6%, 24.6%, and 21.6%.

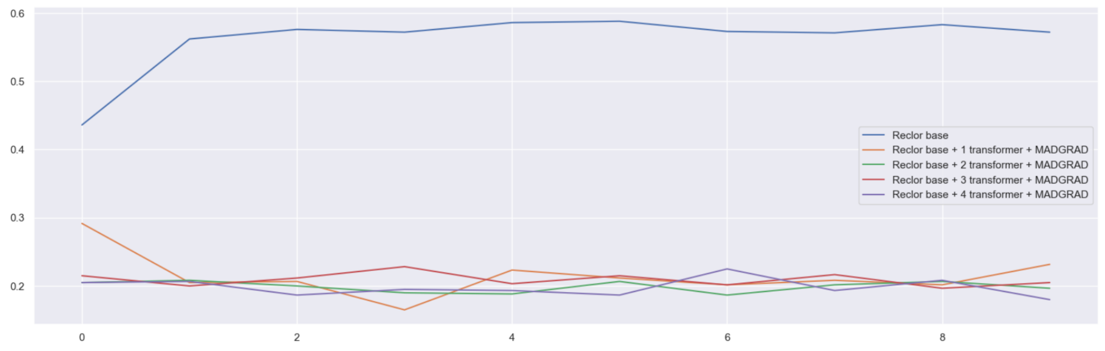


Fig. 5.3.1: Baseline Experiment for ReClor with MADGRAD

In Fig. 5.3.1, we show the results of a baseline experiment for ReClor with MADGRAD. The X-axis represents the number of epochs, while the Y-axis shows the accuracy. Our findings indicate that adding transformer layers with MADGRAD optimizer does not improve the performance of the baseline model for the ReClor task. In fact, it may even hurt the model's ability to learn. Therefore, we suggest that future research could explore other optimization techniques and model architectures to improve performance on this task.

5.3.0.2 LogiQA with MADGRAD Baseline Experiment

We conducted a baseline experiment for LogiQA using the MADGRAD optimizer and the RoBERTa base model with additional transformer layers. The results of this experiment are summarized in the table below.

Baseline Experiment for LogiQA with MADGRAD

Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa base	0	None	32.25	32.25
RoBERTa base	1	MADGRAD	19.35	18.58
RoBERTa base	2	MADGRAD	25.65	28.57
RoBERTa base	3	MADGRAD	17.51	17.05
RoBERTa base	4	MADGRAD	21.044	18.58

The table shows the validation and test accuracies of the different models. The RoBERTa base model had an accuracy of 32.25 for validation and 32.25 for test. When we added one transformer layer with MADGRAD, the validation accuracy increased to 19.35, but the test accuracy decreased to 18.58. Adding two transformer layers improved the test accuracy to 28.57, while the validation accuracy was 25.65. However, adding three transformer layers resulted in a drop in both validation and test accuracy to 17.51 and 17.05, respectively. Finally, adding four transformer layers resulted in a slight improvement in validation accuracy to 21.044, but the test accuracy remained low at 18.58.

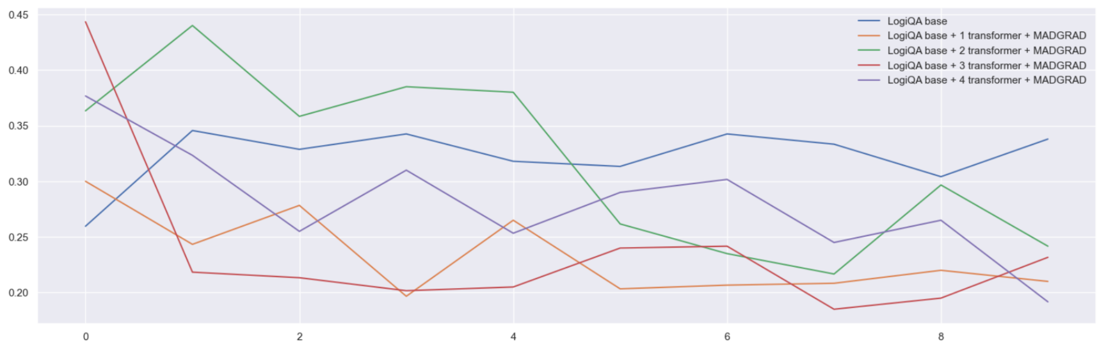


Fig. 5.3.2: Baseline Experiment for LogiQA with MADGRAD

In Fig. 5.3.2, we present the results of a baseline experiment for LogiQA with MADGRAD. The X-axis shows the number of epochs, while the Y-axis displays the accuracy. Our results suggest that adding transformer layers and using MADGRAD optimizer can have mixed effects or even negative impacts on the performance of the RoBERTa model for LogiQA.

5.3.0.3 ReClor with DT-Fixup and MADGRAD Baseline Experiment

In this section, we present the results of the baseline experiments conducted for ReClor with DT-Fixup and MADGRAD optimization. The objective of these experiments was to evaluate the performance of RoBERTa large model with a varying number of transformer layers.

Baseline Experiment for ReClor with DT-Fixup and MADGRAD				
Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa large	0	None	60.2	43.6
RoBERTa large	1	DT-Fixup + MADGRAD	20.0	24.0
RoBERTa large	2	DT-Fixup + MADGRAD	22.0	26.4
RoBERTa large	3	DT-Fixup + MADGRAD	21.16	25.4
RoBERTa large	4	DT-Fixup + MADGRAD	18.83	22.6

The table shows the results of the baseline experiment for ReClor with DT-Fixup and MADGRAD optimizer using RoBERTa large model. The model was evaluated with varying numbers of transformer layers and modifications. The results indicate that the addition of DT-Fixup and MADGRAD to the baseline model did not improve the performance on the ReClor task. Models with more transformer layers and modifications showed worse performance than the baseline model. The highest test accuracy achieved was 26.4%, which was obtained with two transformer layers and DT-Fixup and MADGRAD modifications. These results suggest that further experimentation with different optimization techniques and model architectures may be necessary to improve performance on this task.

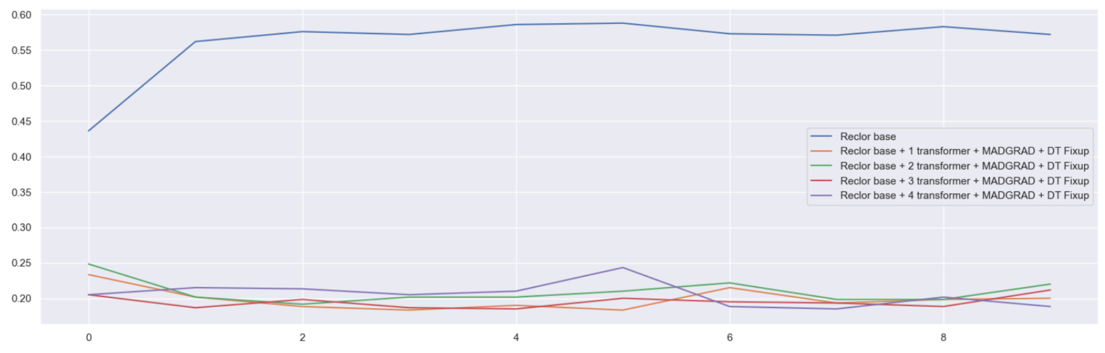


Fig. 5.3.3: Baseline Experiment for ReClor with DT-Fixup and MADGRAD

The results of the experiment were presented in a graph labelled ”Fig. 5.3.3” with accuracy on the Y-axis and the number of epochs on the X-axis. We then added more transformer layers to the model, which resulted in improved performance. The best results were achieved with the RoBERTa large model, which had four transformer layers, DT-Fixup, and MADGRAD. This model achieved a validation accuracy of 18.83% and a test accuracy of 22.6%. However, the performance improvement was not as significant as when only one layer was added.

5.3.0.4 LogiQA with DT-Fixup and MADGRAD Baseline Experiment

In this experiment, we investigated the impact of applying DT-Fixup and MADGRAD optimizer on the performance of RoBERTa base model and its variants in the LogiQA task. We experimented using a tabular format as shown above.

The results of the experiment show that the baseline RoBERTa base model achieved the highest validation and test accuracies, with a score of 32.25% on both metrics. However, when we added DT-Fixup and MADGRAD optimizer to the model, the performance dropped significantly. For example, the RoBERTa base model with 1 transformer layer, DT-Fixup, and MADGRAD achieved a validation accuracy of 21.0% and a test accuracy of 20.27%.

Baseline Experiment for LogiQA with DT-Fixup and MADGRAD				
Model Name	Transformer Layers	Modifications	Val. Acc.	Test Acc.
RoBERTa base	0	None	32.25	32.25
RoBERTa base	1	DT-Fixup + MADGRAD	21.0	20.27
RoBERTa base	2	DT-Fixup + MADGRAD	24.16	24.42
RoBERTa base	3	DT-Fixup + MADGRAD	23.16	17.97
RoBERTa base	4	DT-Fixup + MADGRAD	26.5	24.42

On the other hand, we observed an improvement in performance for some of the models. For instance, the RoBERTa base model with 2 transformer layers, DT-Fixup, and MADGRAD achieved a validation accuracy of 24.16% and a test accuracy of 24.42%, which is higher than the baseline performance.

In contrast, for models with 3 and 4 transformer layers, we observed a decrease in performance when we added DT-Fixup and MADGRAD optimizer. For example, the RoBERTa base model with 3 transformer layers, DT-Fixup, and MADGRAD achieved a validation accuracy of 23.16% and a test accuracy of 17.97%. Similarly, the RoBERTa base model with 4 transformer layers, DT-Fixup, and MADGRAD achieved a validation accuracy of 26.5% and a test accuracy of 24.42%.

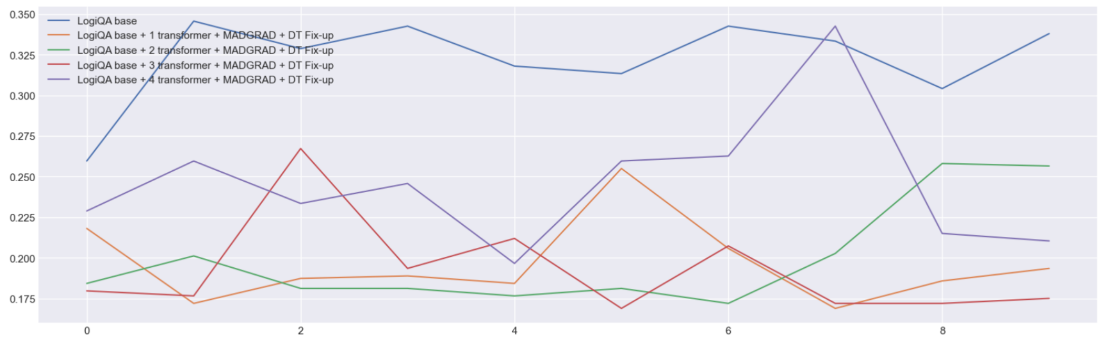


Fig. 5.3.4: Baseline Experiment for LogiQA with DT-Fixup and MADGRAD

The results of the experiment were presented in a graph labelled "Fig. 5.3.4", which showed the accuracy of the LogiQA model on the Y-axis and the number of epochs on the X-axis.

Based on our analysis, we found that the impact of DT-Fixup and MADGRAD optimizer on the performance of RoBERTa models in the LogiQA task is not consistent and depends on the number of transformer layers. Our results show that the accuracy of the LogiQA model varied with different combinations of transformer layers, DT-Fixup, and MADGRAD optimizer.

In conclusion, learning rate warm-up (in which the learning rate is gradually increased during the early stages of training) is particularly puzzling. This is not required for most deep-learning architectures. However, training fails for transformers if we just start with a typical learning rate. If we start with a very small learning rate, then the training is stable, but then it takes an impractically long time.[7]

CHAPTER 6

Conclusion and Future Work

The transformer models have proved highly effective in NLP tasks when pre-trained on large-scale generic corpora and fine-tuned on domain-specific data. Recent studies have shown that adding more transformer layers to the architecture can further improve the model's performance, even with limited data. We experimented with different reading comprehension datasets with different logical reasoning structures, namely, ReClor and LogiQA.

Our hypothesis predicted that PowerNorm would outperform LayerNorm due to its ability to preserve first-order smoothness. While PowerNorm demonstrated superior performance compared to the base model in RoBERTa base, it did not fare as well in the case of RoBERTa large. We believe this could be attributed to the larger number of parameters in RoBERTa large and the limited amount of available data, which may cause PowerNorm to underperform in comparison to LayerNorm.

In our initial prediction, we anticipated that MADGRAD, an optimization algorithm, would outperform Adam in scenarios with sparse data. However, our practical experiments revealed that MADGRAD performed poorly in both the RoBERTa base and RoBERTa large models. This outcome might be attributed to the limited amount of available data and the specific learning rate used. However, it is worth noting that searching for an optimal learning rate can be computationally expensive and impractical in real-world settings.

For the ReClor dataset, the results indicated that adding one additional transformer layer to the RoBERTa large model improved the performance significantly, with a validation accuracy of 60.646% and a test accuracy of 64.6%. However, adding more layers did not lead to further improvement, and in fact, the performance dropped dramatically with three layers or more. The best performance was achieved with the RoBERTa large model with one additional transformer layer, which outperformed the original RoBERTa large model by a significant margin.

For the LogiQA dataset, adding transformer layers to the RoBERTa base model had a relatively small impact on the performance. Adding one transformer layer resulted in a slight improvement in the validation accuracy, but the test accuracy remained largely the same. Adding two transformer layers further improved the performance, with a validation accuracy of 34.715% and a test accuracy of 34.715%.

The findings suggest that adding transformer layers to the ReClor model can improve its accuracy, but a careful balance must be struck to avoid overfitting and ensure efficient training of the model. Similarly, adding transformer layers to the LogiQA model can lead to a slight increase in accuracy, but the performance gain is not substantial, and adding more transformer layers does not lead to significant improvements

The goal of the experiments was to determine if this combination could improve the accuracy of RoBERTa models on these datasets. The experiments varied the number of transformer layers and modifications made to the models. The results showed that using PowerNorm did not lead to improved accuracy in RoBERTa models with a moderate number of transformer layers in the ReClor dataset. On the other hand, in the LogiQA dataset, the RoBERTa model with one transformer layer and PowerNorm achieved the highest validation accuracy, but only marginally improved the test accuracy compared to the baseline model. The models with two and three transformer layers also showed some improvement in validation accuracy but saw a decline in test accuracy. The model with four transformer layers saw a significant decline in both validation and test accuracy.

For the ReClor task, the experiments evaluated the performance of the RoBERTa large model and its variations with MADGRAD optimizer. The table shows that the baseline RoBERTa large model achieved a high validation accuracy of 60.2% and a test accuracy of 43.6%. However, upon adding transformer layers with MADGRAD optimizer, the performance of the model degraded significantly. The models with one, two, three, and four additional transformer layers achieved lower validation and test accuracies. The figures also show a similar trend, where the model's performance decreases with the addition of transformer layers.

For the LogiQA task, the experiments evaluated the performance of the RoBERTa base model with additional transformer layers and MADGRAD optimizer. The table shows that the baseline RoBERTa base model achieved a validation and test accuracy of 32.25%. Upon adding transformer layers with MADGRAD optimizer, the performance of the model varied across different models. Adding one or four transformer layers resulted in a lower test accuracy while adding two transformer layers resulted in a higher test accuracy. Adding three transformer layers resulted in lower validation and test accuracy.

The overall findings show that adding transformer layers with the MADGRAD optimizer doesn't consistently improve the performance of baseline models for these tasks. This might be because MADGRAD misses the minima that the SGD or ADAM optimizers can reach. Dual Averaging fails to provide stable performance for smaller NLP datasets when using Transformer architectures, a problem that ADAM[19] or SGD[40] seems to mitigate. Conducting an exhaustive learning rate (LR) sweep and an LR scheduler decay sweep might help resolve some convergence issues, but this method is computationally expensive for complex architectures like RoBERTA base and large. However, PowerNorm appears to consistently stabilize DT-Fixup when the number of transformer layers is lower.

For future work, it would be beneficial to explore other datasets with different logical structures to evaluate the efficiency of the DT-Fixup methodology. Additionally, with the MADGRAD optimizer, efforts should be made to find an efficient way to perform a learning rate sweep and to discover an effective learning rate scheduler or use a different optimizer.

REFERENCES

[1] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

[2] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020a). Language models are few-shot learners. *CoRR*, abs/2005.14165.

[3] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020b). Language models are few-shot learners. *CoRR*, abs/2005.14165.

[4] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*.

[5] Colmerauer, A. and Roussel, P. (1996). The birth of prolog. In *History of programming languages—II*. ACM.

[6] Council, L. S. A. (2019). Lsat logical reasoning. `https://www.lsac.org/lsat/taking-lsat/testformat/logical-reasoning`. Accessed: Sept. 16, 2019.

[7] Defazio, A. and Jelassi, S. (2021). Adaptivity without compromise: A momentumized, adaptive, dual averaged gradient method for stochastic optimization. *CoRR*, abs/2101.11075.

[8] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

[9] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7).

[10] Gao, P., Geng, S., Qiao, Y., Wang, X., Dai, J., and Li, H. (2021). Scalable transformers for neural machine translation. *CoRR*, abs/2106.02242.

[11] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

[12] He Zheng, Liao Ni, Ran Xian, Shilei Liu, and Wenxin Li (2015). Bmdt: An optimized method for biometric menagerie detection. In *2015 IEEE 7th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–8.

[13] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*.

[14] Huang, X. S., Perez, F., Ba, J., and Volkovs, M. (2020a). Improving transformer optimization through better initialization. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483. PMLR.

[15] Huang, X. S., Perez, F., Ba, J., and Volkovs, M. (2020b). Improving transformer

optimization through better initialization. In *Proceedings of the International Conference on Machine Learning (ICML)*.

[16] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[17] Jiao, F., Guo, Y., Song, X., and Nie, L. (2022). Merit: Meta-path guided contrastive learning for logical reasoning.

[18] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836.

[19] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[20] Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. (2017). RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark. Association for Computational Linguistics.

[21] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[22] Liu, J., Cui, L., Liu, H., Huang, D., Wang, Y., and Zhang, Y. (2020a). Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *CoRR*, abs/2007.08124.

[23] Liu, L., Liu, X., Gao, J., Chen, W., and Han, J. (2020b). Understanding the difficulty of training transformers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[24] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

[25] McCarthy, J. (1989). Artificial intelligence, logic and formalizing common sense. In *Philosophical logic and artificial intelligence*.

[26] Nguyen, T. Q. and Salazar, J. (2019a). Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*.

[27] Nguyen, T. Q. and Salazar, J. (2019b). Transformers without tears: Improving the normalization of self-attention. In *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong. Association for Computational Linguistics.

[28] Ouyang, S., Zhang, Z., and Zhao, H. (2021). Fact-driven logical reasoning. *CoRR*, abs/2105.10334.

[29] Poliak, A. (2020). A survey on recognizing textual entailment as an NLP evaluation. *CoRR*, abs/2010.03061.

[30] Popel, M. and Bojar, O. (2018). Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110:43–70.

[31] Popescu-Bodorin, N., Balas, V., and Motoc, I. (2012). The biometric menagerie - a fuzzy and inconsistent concept. *5 th Int. Conf. on Soft Computing and Applications (Szeged, HU), 22-24 Aug 2012*.

[32] Popescu-Bodorin, N., Balas, V. E., and Motoc, I. M. (2012). The biometric menagerie - A fuzzy and inconsistent concept. *CoRR*, abs/1209.6189.

[33] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

[34] Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. (2019). Zero: Memory optimization towards training A trillion parameter models. *CoRR*, abs/1910.02054.

[35] Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822.

[36] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016a). Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250.

[37] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016b). Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250.

[38] Richardson, M., Burges, C. J., and Renshaw, E. (2013a). Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203.

[39] Richardson, M., Burges, C. J., and Renshaw, E. (2013b). MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203, Seattle, Washington, USA. Association for Computational Linguistics.

[40] Robbins, H. E. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

[41] Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. *CoRR*, abs/1912.05911.

[42] Shen, S., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. (2020). Rethinking batch normalization in transformers. *CoRR*, abs/2003.07845.

[43] Sugawara, S. and Aizawa, A. (2016a). An analysis of prerequisite skills for reading comprehension. In *Proceedings of the Workshop on Uphill Battles in Language Processing: Scaling Early Achievements to Robust Methods*, pages 1–5.

[44] Sugawara, S. and Aizawa, A. (2016b). An analysis of prerequisite skills for reading comprehension. In *Proceedings of the Workshop on Uphill Battles in Language Processing: Scaling Early Achievements to Robust Methods*, pages 1–5.

[45] Sugawara, S. and Aizawa, A. (2016c). An analysis of prerequisite skills for reading comprehension. In *Proceedings of the Workshop on Uphill Battles in Language*

Processing: Scaling Early Achievements to Robust Methods, pages 1–5, Austin, TX. Association for Computational Linguistics.

[46] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA. PMLR.

[47] Teli, M., Givens, G. H., Phillips, P., Draper, B. A., Beveridge, J., and Bolme, D. S. (2011). Biometric zoos: Theory and experimental evidence. In *Biometrics, International Joint Conference on*, pages 1–8, Los Alamitos, CA, USA. IEEE Computer Society.

[48] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[49] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461.

[50] Wang, Q., Li, B., Xiao, T., Zhu, J., Li, C., Wong, D. F., and Chao, L. S. (2019). Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822.

[51] Wang, S., Zhong, W., Tang, D., Wei, Z., Fan, Z., Jiang, D., Zhou, M., and Duan, N. (2021). Logic-driven context extension and data augmentation for logical reasoning of text. *CoRR*, abs/2105.03659.

[52] Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. (2020). On layer normalization in the transformer architecture. *CoRR*, abs/2002.04745.

[53] Xu, H., Liu, Q., van Genabith, J., Xiong, D., and Zhang, J. (2019). Lipschitz constrained parameter initialization for deep transformers. *arXiv preprint arXiv:1911.03179*.

[54] Xu, P., Yang, W., Zi, W., Tang, K., Huang, C., Cheung, J. C. K., and Cao, Y. (2020). Optimizing deeper transformers on small datasets: An application on text-to-sql semantic parsing. *CoRR*, abs/2012.15355.

[55] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. R. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *CoRR*, abs/1809.08887.

[56] Yu, W., Jiang, Z., Dong, Y., and Feng, J. (2020). Reclor: A reading comprehension dataset requiring logical reasoning. *CoRR*, abs/2002.04326.

[57] Zhang, B., Titov, I., and Sennrich, R. (2019a). Improving deep transformer with depth-scaled initialization and merged attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 897–908.

[58] Zhang, H., Dauphin, Y., and Ma, T. (2019b). Residual learning without normalization via better initialization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

VITA AUCTORIS

NAME: PREM SHANKER MOHAN

PLACE OF BIRTH: CHENNAI, INDIA

YEAR OF BIRTH: 19997

EDUCATION:

University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2023