

**COMBINATION OF GENERATIVE
ARTIFICIAL INTELLIGENCE AND DEEP
REINFORCEMENT LEARNING:
PERFORMANCE COMPARISON**

LIM FANG NIE

UNIVERSITI TUNKU ABDUL RAHMAN

**COMBINATION OF GENERATIVE ARTIFICIAL INTELLIGENCE
AND DEEP REINFORCEMENT LEARNING: PERFORMANCE
COMPARISON**

Lim Fang Nie

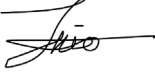
**A project report submitted in partial fulfilment of
the requirements for the award of Bachelor of
Science (Honours) Software Engineering**

**Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

September 2024

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : 

Name : LIM FANG NIE

ID No. : 2200481

Date : 12/9/2024

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**COMBINATION OF GENERATIVE ARTIFICIAL INTELLIGENCE AND DEEP REINFORCEMENT LEARNING: PERFORMANCE COMPARISON**” was prepared by **LIM FANG NIE** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Software Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Yau Kok Lim

Date : 4 October 2024

Signature : _____
Co-Supervisor : _____
Date : _____

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2024, Lim Fang Nie. All right reserved.

ABSTRACT

In this study, we explore the integration of Generative Adversarial Networks (GANs) and Deep Reinforcement Learning (DRL) methods, focusing on the performance comparison between different architectures of Sequence Generative Adversarial Networks (SeqGAN) and policy gradient algorithms. We address key challenges in text generation, such as maintaining narrative coherence over long sequences, reducing text repetition, and optimizing SeqGAN for diverse textual outputs. The study incorporates architectural innovations like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) that enhance the ability of SeqGAN to capture long-range dependencies in sequences, while attention mechanisms improve contextual awareness by selectively focusing on relevant parts of the sequence. Through extensive experiments, we analyze the influence of various neural network configurations and regulatory mechanisms, including gradient penalties and regularization on the quality of the generated text. Our findings show a 15% increase in BLEU scores, highlighting significant improvements in text coherence and diversity across various datasets, demonstrating the effectiveness of integrating SeqGAN with policy gradient methods for automated content generation.

TABLE OF CONTENTS

TABLE OF CONTENTS		i
LIST OF TABLES		iv
LIST OF FIGURES		v
LIST OF ALGORITHMS		viii
LIST OF SYMBOLS / ABBREVIATIONS		1
 CHAPTER		
1	INTRODUCTION	2
1.1	General Introduction	2
1.1.1	Generative Adversarial Networks (GANs)	2
1.1.2	Deep Reinforcement Learning (DRL)	3
1.1.3	SeqGAN with Policy Gradient	4
1.2	Importance of the Study	8
1.3	Problem Statement	9
1.3.1	Optimization Challenges in SeqGAN	9
1.3.2	Repetition and Lack of Text Coherence	9
1.3.3	Lack of a Comprehensive Integration Framework	10
1.4	Aim and Objectives	10
1.5	Scope and Limitation of the Study	11
2	LITERATURE REVIEW	13
2.1	Introduction	13
2.2	Generative Adversarial Networks and SeqGAN: Comparative Overview	14
2.2.1	Generative Adversarial Networks (GAN)	14
2.2.2	Sequence Generative Adversarial Nets	17
2.3	Comparative Analysis of Policy Gradient Methods	22
2.3.1	Policy Gradient Algorithms	22
2.4	Architecture Innovation	30
2.4.1	Long short-term memory (LSTM)	31

2.4.2	Gated Recurrent Unit (GRU)	33
2.4.3	Attention Mechanisms	35
2.4.4	Conditional GANs	36
2.5	Evaluation Metric	40
2.6	Regulatory Mechanisms and Penalties	44
2.6.1	Gradient Penalty	46
2.6.2	L1/L2 Regularization	47
2.6.3	Entropy Penalty	48
2.6.4	Semantic Consistency Penalty	49
2.7	Summary	54
3	METHODOLOGY AND WORK PLAN	56
3.1	Introduction	56
3.2	Software and Tools	58
3.2.1	Tools	58
3.2.2	Hardware Environment	58
3.2.3	Software Environment	59
3.3	Work Plan	61
4	PROJECT INITIAL SPECIFICATION	64
4.1	Introduction	64
4.2	Data Collection	64
4.3	Data Processing	64
4.4	SeqGAN Model Enhancement	66
4.4.1	GRU Layer Integration	66
4.4.2	Transformer Intergration	66
4.4.3	Reward Structure	67
4.5	Optimization of SeqGAN	67
4.6	Evaluation Metrics and Performance Analysis	68
4.7	Summary	69
5	RESULT AND DISCUSSION	70
5.1	System Performance	70
5.1.1	Quantitative Metric	70
5.1.2	Evaluation of Poem	82
5.2	System Demonstrations	86
5.2.1	Model Enhancement	86

6	CONCLUSIONS AND RECOMMENDATIONS	97
6.1	Conclusion	97
6.2	Recommendations for future work	97
	REFERENCES	100

LIST OF TABLES

Table 2.1:	Feature Comparison of GAN and SeqGAN	21
Table 2.2:	Training Process Comparison of GAN and SeqGAN	21
Table 2.3:	Comparison of Policy Gradient Algorithms	29
Table 2.4:	Sustainability for SeqGAN and Test Generation	30
Table 2.5:	Comparisons of SeqGAN Architectures Innovation	37
Table 2.6:	Comparison of Training Techniques for SeqGAN	39
Table 2.7:	Comparison of various Evaluation Metrics	43
Table 2.8:	Comparison of L1 and L2 regularization	48
Table 2.9:	Comparison of various regularization and penalty techniques	51
Table 2.10:	Implication of Regulatory Mechanisms on Model Performance	53
Table 3.1:	Hardware specifications	59
Table 3.2:	Software specifications	60
Table 4.1:	Overview of the Proposed Solution	69
Table 5.1:	Hyperparameter configurations	70
Table 5.2:	Generator Adversarial Training Loss Result	71
Table 5.3:	Discriminator Adversarial Training Loss Result	73
Table 5.4:	Generator Oracal NLL result	75
Table 5.5:	Discriminator Oracal NLL result	77
Table 5.6:	BLEU Score results for both model	79

LIST OF FIGURES

Figure 1.1:	Structure of DRL	3
Figure 2.1:	Structure of GANs	15
Figure 2.2:	Architecture of Multilayer Perceptrons (MLPs)	16
Figure 2.3:	Structure of SeqGANs	17
Figure 2.4:	Monte Carlo Tree Search (MCTS) process	18
Figure 2.5:	Architecture of Encoder-Decoder Network	19
Figure 2.6:	Unit of RNN and LSTM	31
Figure 2.7:	Unit of GRU	34
Figure 2.8:	Wasserstein GAN with a gradient penalty for Length of Stay ⁴⁶	
Figure 3.1:	Summary of Project Workflow	56
Figure 3.2:	Project Work Breakdown Structure (WBS)	62
Figure 3.3:	Gantt Chart	63
Figure 4.1:	Sample of Chinese Poetry	64
Figure 4.2:	Sample of Chinese Poetry after tokenization	65
Figure 5.1:	Generator loss of baseline model using setting 1 and setting 2	72
Figure 5.2:	Generator loss of enhanced model using setting 1 and setting 2	72
Figure 5.3:	Discriminator loss of baseline model using setting 1 and setting 2	73
Figure 5.4:	Discriminator loss of enhanced model using setting 1 and setting 2	74
Figure 5.5:	Generator Oracle NLL of baseline model using setting 1 and setting 2	75

Figure 5.6:	Generator Oracle NLL of enhanced model using setting 1 and setting 2	76
Figure 5.7:	Discriminator Oracle NLL of baseline model using setting 1 and setting 2	77
Figure 5.8 :	Discriminator Oracle NLL of enhanced model using setting 1 and setting 2	78
Figure 5.9:	BLEU Scores Comparison between Baseline and Enhanced Model	79
Figure 5.11:	Generated poems using enhanced model	83
Figure 5.10	: Generated poems using baseline model	83
Figure 5.12	: Climbing White Stork Tower from Wang Zhihuan	84
Figure 5.13	: Sample lines poems from the baseline model	84
Figure 5.14	: Sample lines poems for enhanced model	85
Figure 5.15	: Overview of TargetGRU class	86
Figure 5.16	: Embedding Layer	87
Figure 5.17	: GRU Layer	87
Figure 5.18	: Fully Connected Layer and Softmax	87
Figure 5.19	: Forward Pass	88
Figure 5.20	: Overview of TargetGRU class	88
Figure 5.21	: Step Method for Token-by-Token Processing	88
Figure 5.22	: init_hidden function	89
Figure 5.23	: init_params function	89
Figure 5.24	: Architecture of Transformer	90
Figure 5.25	: PositionEncoding method	91
Figure 5.26	: Positional Encoding Matrix	91
Figure 5.27	: Division Term	92
Figure 5.28	: Sine and Cosine Functions	92

Figure 5.29	: Unsqueeze and Register Buffer	92
Figure 5.30	: Forward Pass Function	93
Figure 5.31	: Function of Gradient Penalty Calculation	94
Figure 5.32	: Interpolation Between Real and Fake Samples	94
Figure 5.33	: Discriminator Output on Interpolated Samples	95
Figure 5.34	: Gradient Computation	95
Figure 5.35	: Reshape Gradients and Gradient Norm	95
Figure 5.36	: Gradient Penalty calculation formation	96

LIST OF ALGORITHMS

Algorithm 1.1:	The algorithm framework of SeqGAN with policy gradient	5
Algorithm 2.1:	Algorithm of REINFORCE	24
Algorithm 2.2:	Algorithm of TRPO	26
Algorithm 2.3: :	Algorithm of PPO	28

LIST OF SYMBOLS / ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
cGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Networks
CVPR	Computer Vision and Pattern Recognition
DRL	Deep Reinforcement Learning
GAN	Generative Adversarial Network
GRU	Gated Recurrent Units
GP	Gradient Penalty
IDE	Integrated Development Environment
LSTM	Long Short-Term Memory
MCTS	Monte Carlo Tree Search
MLE	Maximum Likelihood Estimation
MLP	Multilayer Perceptrons
NLL	Negative Log-Likelihood
NLP	Natural Language Processing
PPO	Proximal Policy Optimization
RNN	Recurrent Neural Network
SeqGAN	Sequence Generative Adversarial Network
TRPO	Trust Region Policy Optimization
WBS	Work Breakdown Structure
WGAN	Wasserstein Generative Adversarial Network
WGAN-GP	Wasserstein Loss with Gradient Penalty

CHAPTER 1

INTRODUCTION

1.1 General Introduction

In recent years, artificial intelligence has been buzzing with remarkable advancements in the field of generative artificial intelligence and deep reinforcement learning (DRL). Generative artificial intelligence, known as Generative AI, is a voluntary field of AI that focuses on generating new data such as images, text, or music. The newly generated data aims to be realistic and true instead of just copies of data. With this approach, machines are allowed to move beyond to analyze existing information and enhance it with the ability to generate new content with remarkable creativity. One of the popular generative AI approaches is generative adversarial networks.

1.1.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are an innovation in the field of Generative AI. This class of machine learning frameworks emerged from the work of Ian Goodfellow and his colleagues in 2014 (Goodfellow et al., 2014). GANs work by pitting two neural networks against each other in a competitive setting. One network is called the generator and another one is called the discriminator. The generator produces realistic data, while the discriminator attempts to differentiate between real and generated fake data. This adversarial process pushes both networks to improve and allows the generator to create realistic sample outputs. Neural networks in machine learning are also referred to as artificial neural networks (ANNs) because of the design that is specifically built to replicate the structure and function of the human brain.

An outstanding case in the world of GANs is StyleGAN2, introduced at the Conference on Computer Vision and Pattern Recognition (CVPR) in 2020. This model leverages transfer learning to produce a virtually infinite number of portraits with an astounding range of artistic styles (Esser et al., 2020). It allows for fine-grained control over specific details, such as facial expressions and poses.

1.1.2 Deep Reinforcement Learning (DRL)

While Deep Reinforcement Learning (DRL) is an area of artificial intelligence that combines reinforcement and deep learning. It primarily focuses on training agents to make optimal decisions in an environment through trial and error. DRL agents differ from classical supervised learning, which is supplied with labelled examples. They learn by interacting with the environment and receive rewards for desired behaviours. This allows the agent to learn complex strategies and adapt themselves to dynamic situations or scenarios. An exemplary example of DRL would be AlphaGo, a program developed by DeepMind that achieved mastery in the complex game of Go (Silver et al., 2016). AlphaGo, along with its successors, leverages a Monte Carlo tree search algorithm to identify optimal moves. This selection process builds upon knowledge previously acquired through extensive machine learning. Specifically, an artificial neural network, a core component of deep learning, is trained on a huge dataset of human and computer Go games (Silver et al., 2016). The neural network progressively refines its ability to identify the most effective moves and their corresponding winning probabilities. As a result, the tree search algorithm is continuously strengthened, leading to a more sophisticated selection of moves in subsequent iterations.

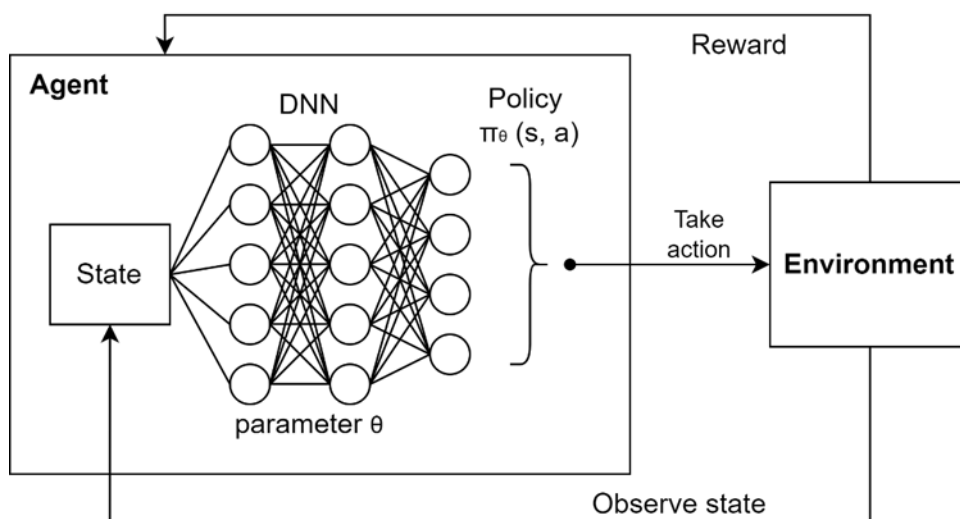


Figure 1.1 demonstrates an agent with a neural network policy taking actions in an environment based on observed states, receiving rewards, and

updating its policy parameters to learn an optimal behaviour over multiple iterations.

Artificial Intelligence has moved past its usual techniques, adopting new strategies that combine the power of generating content with the learning ability of deep reinforcement learning. This research project focuses on investigating the potential benefits of the combination of SeqGAN, a generative AI model, with various policy gradient DRL methods. The potential benefits include improving text quality and coherence, enhancing the ability to generate diverse and novel content, increasing efficiency in training and content generation, and improving the ability to handle long-range dependencies in text. By analyzing these combined models' performance, architecture, neural networks, and key parameters, this project aims to understand the factors influencing their effectiveness and performance. The factors may be the choice of policy gradient algorithm, the architecture of generator and discriminator networks, reward function design, the impact of different pre-training strategies for the generator, and the effect of batch size and sequence length on model performance.

1.1.3 SeqGAN with Policy Gradient

In addressing the complexities of text generation, especially in generating coherent content, the integration of SeqGAN with policy gradient methods. This methodology leverages the adversarial framework established by Goodfellow et al. (2014) and adapts it using reinforcement learning (RL) strategies to optimize the generative process over extended sequences without relying on intermediate rewards (Yu, et al., 2017; Sutton & Barto, 2018).

Algorithm 1: SeqGAN

Require: Discriminator, D_ϕ ; Generator policy, G_θ ; roll-out policy, G_β a

sequence data $S = \{X_{1:T}\}$

01: Initialize G_θ , D_ϕ with random weights θ , ϕ .

02: Pre-train G_θ using MLE on S

03: $\beta \leftarrow \theta$

```

04: Generate negative samples using  $G_\theta$  for training  $D_\phi$ 
05: Pre-train  $D_\phi$  via minimizing the cross entropy
06: repeat
07:   for g-steps do
08:     Generate a sequence  $Y_{1:T} = (y_1, \dots, y_t, \dots, y_T) \sim G_\theta$ 
09:     for  $t$  in  $1:T$  do
10:       Calculate  $Q(a = y_t; s = Y_{1:t-1})$  by Eq. (1.1)
11:     end for
12:     Update generator settings using policy gradients.
    Eq.(1.3)
13:   end for
14:   for d-steps do
15:     Use current  $G_\theta$  to generate negative examples and
    combine with specified positive examples,  $S$ 
16:     Train discriminator  $D_\phi$  for  $k$  epochs using Eq. (1.4)
17:   end for
18:    $\beta \leftarrow \theta$ 
19: until SeqGAN converges

```

Algorithm 1.1: The algorithm framework of SeqGAN with policy gradient

The algorithm for integrating SeqGAN with policy gradient methods is shown in Algorithm 1.1. In the initialization phase, the step involves initializing the generator model G_θ , discriminator model D_ϕ , and roll-out policy G_β with stochastic weights. This setup employs sequence data $S = \{X_{1:T}\}$ for training, aligning with the preparatory requirements outlined by Yu et al. (2017). Sequentially, G_θ undergoes pre-training utilizing Maximum Likelihood Estimation (MLE) on S , paralleled by the discriminator D_ϕ 's training to minimize cross-entropy, thus enhancing its capability to differentiate between genuine and synthetically generated sequences. The roll-up policy is employed to complete the partial sequence. The Q-function is defined as:

$$Q_{D_\phi}^{G_\theta}(s = Y_{1:t-1}, a = y_t)$$

$$= \begin{cases} \frac{1}{N} \sum_{n=1}^N D_\phi(Y_{1:T}^n), & Y_{1:T}^n \in MC^{G_\beta}(Y_{1:t}; N) \text{ for } t < T \\ D_\phi(Y_{1:t}) & \text{for } t = T \end{cases} \quad (1.1)$$

MC refers to the sampling set as follows:

$$\{Y_{1:T}^1\}, \dots, \{Y_{1:T}^N\} = MC^{G_\beta}(Y_{1:t}; N) \quad (1.2)$$

The standard policy gradient is as follows:

$$\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta) \quad (1.3)$$

The discriminator loss is as follows:

$$\min_{\phi} - E_{Y \sim P_{data}} [\log D_\phi(Y)] - E_{Y \sim G_\theta} [\log(1 - D_\phi(Y))] \quad (1.4)$$

In the pre-training stage of the process, the generator G_θ undergoes preliminary training by using MLE based on the sequence data S . Concurrently, the discriminator, D_ϕ is trained to minimize cross-entropy, enhancing its ability to distinguish between genuine and artificially generated sequences. This dual training approach is foundational, setting the stage for the more subtle adversarial training dynamics that follow.

During the adversarial training phase, the generator begins a series of refinement steps, known as G-steps. In each iteration, it crafts sequences $Y_{1:T} = (y_1, \dots, y_T)$. with every timestep t calculated through the action-value function $Q(a = y_t; s = y_{1:t-1})$. This function aims to predict the expected rewards for actions y_t within the context of the sequences generated thus far. The primary goal during these steps is to adjust the generator's parameters (θ) via policy gradient methods, thereby maximizing the expected rewards for the generated narratives. This methodical refinement ensures that the generator learns to produce sequences that are not only coherent but also align closely with the desired outcomes.

Simultaneously, the discriminator undergoes optimization through D-steps. It utilizes the currently refined G_θ to generate negative samples, which are then mixed with positive examples from S . This mix is used to train $D\phi$ over several epochs, significantly improving its discriminative power. By accurately identifying genuine from generated sequences, $D\phi$ provides critical feedback that informs further refinements to G_θ .

The iterative process of adversarial training continues until a convergence condition is met. This point of convergence is characterized by the generator's ability to produce sequences that the discriminator cannot easily distinguish from real data. Achieving this milestone signifies a successful adversarial training process, indicating that the generator and discriminator models have been finely tuned to work in tandem, producing high-quality, realistic sequences. This marks a crucial step forward in the development of generative models, pushing the boundaries of what's possible with artificial sequence generation.

The integration of SeqGAN with policy gradient methods represents a significant stride towards solving the challenges of text generation in generative AI. Through this novel approach, this research not only contributes to the theoretical understanding of generative models and reinforcement learning but also paves the way for new applications and improvements in AI-driven text generation technologies.

The research aims to utilize reinforcement learning to directly optimize the sequence generation process and focus on the end goal of producing coherent and contextually rich text. This method addresses the inherent limitations of traditional SeqGAN by enhancing the model's ability to maintain narrative consistency over longer sequences. It also provides a framework for fine-tuning generative models based on holistic sequence quality, rather than immediate next-token predictions.

1.2 Importance of the Study

Generative artificial intelligence has reached a new peak in innovation in machine learning with the combination of deep reinforcement learning mechanisms. In generative AI, Sequence Generative Adversarial Networks (SeqGANs) have played a role in text sequence generation due to their capability to generate text sequences that reflect human-like coherence. (Yu, et al., 2017). While successful in generating shorter forms of text, SeqGANs struggle to generate longer forms of text as their performance declines with long-term contextual dependencies. (Holtzman, et al., 2019)

Due to the well-established learning mechanism that involves trial and error, policy gradient methods have a better chance to help SeqGANs improve their coherence and text length. (Espeholt, et al., 2018; Sutton & Barto, 2018). However, there is little work done to understand how the policy gradient methods are compatible with different SeqGAN architectures for different categories of text generation tasks. (Liu, et al., 2020)

This research aims to fill this gap by not only exploring and evaluating the compatibility of different combinations of SeqGAN architectures but to also provide a general understanding of their dynamics in text generation which is because of fundamental importance for the optimization of many text generation tasks. From a commercial angle, this study can be implemented to enhance content writing, marketing, education, and customer service through automated content generation.

Therefore, this research is not only about discovering the opportunities for integrating SeqGAN and DRL but is also an essential possibility to take an initial step to develop the use of generative AI to the whole extent of text creation. The expected outcomes of this study show how much impact this work may have on the development of natural language processing and dialogue systems.

1.3 Problem Statement

1.3.1 Optimization Challenges in SeqGAN

The advent of Sequence Generative Adversarial Networks (SeqGANs) has significantly improved text generation that approximates human-level coherence and creativity. However, it has faced some challenges that hinder its performance as the applications expand. The training of SeqGAN is inherently unstable due to the adversarial nature of the process. The generator aims to create sequences that are unidentifiable from real sample, while the discriminator tries to identify the real from the generated sequence correctly. This adversarial training can lead to oscillations, where the generator and discriminator unable to converge, causing the model to not improve or even worsen over time. Besides that, when the generator learns to produce only a few variations of sequences the discriminator cannot easily identify as fake which will cause the model to collapse. As a result, the generated text will lack diversity. This will cause it to produce repetitive and similar outputs which reduce the quality of the generated content.

Krivosheev et al. (2021) highlighted the critical impact of batch size on SeqGAN performance, revealing a delicate balance between computational efficiency and the quality of generated text. Smaller batches tend to produce more diverse text but can increase training instability. In contrast, larger batch sizes may lead to faster convergence but often result in reduced diversity, leading to repetitive and homogeneous outputs. This points to broader optimization challenges within SeqGAN frameworks that significantly affect the quality and utility of generated text.

1.3.2 Repetition and Lack of Text Coherence

SeqGAN often struggles with producing coherent and non-repetitive text, especially for longer sequences. During training, the generator might fall into patterns that produce repetitive text, which makes the output less interesting and engaging. This repetition occurs because the generator might find it easier to generate familiar patterns that the discriminator has previously failed to identify as fake.

Besides that, generating long-form text with a consistent narrative is particularly challenging. SeqGAN can generate short and coherent sequences. However, as the text length increases, it is difficult to maintain a logical flow and context. The model often loses track of the narrative, resulting in disjointed and incoherent text. Lagutin et al. (2021) demonstrated that policy gradient reinforcement learning could refine text generation processes, reducing repetition and improving coherence.

1.3.3 Lack of a Comprehensive Integration Framework

Despite these advancements, a comprehensive framework for integrating SeqGANs with DRL, while leveraging insights from continuous adversarial learning and semantic-enhanced representation is currently lacking. Combining the strengths of adversarial learning (GANs) and reinforcement learning (policy gradients) in a unified framework remains an underexplored area. Developing a robust methodology to seamlessly integrate these techniques is essential for improving the performance of SeqGAN models. There is a need for a structured approach to evaluate different combinations of SeqGAN architectures and policy gradient methods. This involves systematically analyzing the interaction dynamics between various architectures and DRL algorithms to identify the most effective configurations for specific text generation tasks.

Zhang et al. (2022) provide valuable insights into adversarial learning in continuous text feature space and suggest several pathways for improving SeqGAN architectures, including the use of adversarial feature matching to align real and generated text distributions, mitigating high-variance gradient estimations for stable training, and integrating transformer models to better capture long-range dependencies and contextual information. However, these enhancements need to be integrated and evaluated within a comprehensive framework to determine their effectiveness in practical applications.

1.4 Aim and Objectives

This research project seeks to compare the performance of different combinations formed by combining SeqGAN with various policy gradient methods. There are four objectives have been set and presented below:

- i. To explore the cooperation effects between various SeqGAN architectures and policy gradient methods, aims to improve narrative coherence measured by BLEU scores and custom narrative consistency indices. This includes mapping the dynamics interaction between the two categories to identify those that improve narrative coherence and contextuality by a significant margin. It is essential to align with the project's overall objective of building more sophisticated generative text models.
- ii. To analyze how different SeqGAN architectures affect the coherence of generated textual content when integrated with policy gradient method, influence the coherence of generated textual content. The parameters will be coherence metrics, such as BLEU scores, as well as the development of a narrative consistency index. This assessment will cut across most types of content, such as scripts, articles, and standalone narratives.
- iii. To explore the impact of neural network configurations within SeqGAN models optimised with policy gradient methods, focusing on variables, for instance, the number of hidden layers, activation functions, and overall network dimensions. The goal is to identify optimal configurations that will fully capture on the ability to capture long-range dependencies, narrative structure, and contextual relevance.

1.5 Scope and Limitation of the Study

The scope of this study exceeds the fundamental integration of policy gradient methods with SeqGAN architectures to improve text generation. In particular, it is broadened by multiple findings associated with the investigated regulatory mechanisms and penalties. This research aims to understand how different regulatory strategies affect the learning process, particularly in terms of model adherence to narrative coherence and the mitigation of common generative text issues such as repetition and divergence from context.

Apart from computational experiments on SeqGAN and the performance of policy gradient methods, this research applies a systemic approach to testing the impact of various regulators and penalties, including “reward shaping”, “regularization”, and custom-designed penalty functions that are created to improve text generation. To measure the effectiveness of regulatory mechanisms, both qualitative and quantitative metrics will be introduced and tested across different datasets, such as BLEU scores for coherence and custom narrative consistency indices.

The exploration into regulators and penalties was assumed as a hypothesis that regulated work and specially calibrated constraints can significantly boost the generative capacity of SeqGAN models, especially in terms of producing coherent contextually rich texts. This part of the research is particularly focused on identifying strategies that can simultaneously advance text quality and models’ ability to perform while staying adaptive to many forms of text.

Although investigating regulations and penalties might improve SeqGAN's text generation skills, it complicates the process of designing and optimizing the model. It increases computational costs and risks over-regulation, potentially harming the model's creativity and leading to repetitive outputs. Moreover, finding the optimal balance between regulatory constraints and creative freedom remains a challenging commitment, potentially affecting the scalability of proposed solutions.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In recent years, Artificial intelligence (AI) has undergone a resurgence in the development of advanced machine-learning models capable of producing especially in human-like writing. Generative Adversarial Networks (GANs) like Sequence Generative Adversarial Networks (SeqGANs) have emerged as a powerful tool in the AI toolkit. It possesses the ability to produce content that often eliminates the boundary between human and machine-generated text. This literature review explores SeqGANs and their integration with Deep Reinforcement Learning (DRL) techniques to push the envelope in text generation. DRL that will be focusing in this literature review will be the policy gradient methods

Despite advances in natural language processing (NLP), text generation with consistent quality remains a key research problem such as preserving coherence, maintaining thematic focus, and ensuring novelty across extended outputs are significant hurdles. Combining Sequence Generative Adversarial Networks (SeqGANs) with policy gradient techniques offers a potential solution to improve the capacity of language models in content generation.

This chapter provides a comparative overview of the developments in GANs and SeqGANs, underscoring the evolutionary strides made from their inception to their current applications. A critical analysis of policy gradient methods will illuminate their role and potential in refining SeqGANs, while a discussion on hyperparameter tuning will unravel strategies to optimize these complex models effectively. Innovations in model architecture, evaluation metrics, and regulatory mechanisms are examined through the lens of comparative synthesis to distil their effectiveness in enhancing text generation.

Moreover, the literature review identifies key research gaps and opportunities that have surfaced amidst these advancements. By systematically dissecting and juxtaposing seminal works and recent innovations, this review sets the stage for subsequent chapters, where a comprehensive methodology is designed to explore, test, and possibly transcend the existing boundaries of generative AI.

As AI continues to weave itself into the fabric of digital society, understanding its potential and limitations in creating coherent and extensive narratives is more than an academic pursuit; it is a step towards harnessing AI's full potential in transforming how we produce and interact with textual content across various sectors.

2.2 Generative Adversarial Networks and SeqGAN: Comparative Overview

Generative Adversarial Networks (GANs) and Sequence Generative Adversarial Networks (SeqGANs) are innovative frameworks in the field of machine learning that address different aspects of generative modelling. This section provides an overview of both GANs and SeqGANs, examining their architectures, methodologies, and comparative strengths and weaknesses.

2.2.1 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN), first proposed by Goodfellow et al. (2014), have rapidly evolved into one of the most successful generative models in the field of machine learning. GAN consists of two neural networks: a generator and a discriminator. Both of them are trained concurrently using adversarial methods. The generator learns to create data that is identical to and indistinguishable from genuine data, while the discriminator improves their capacity to discriminate between them.

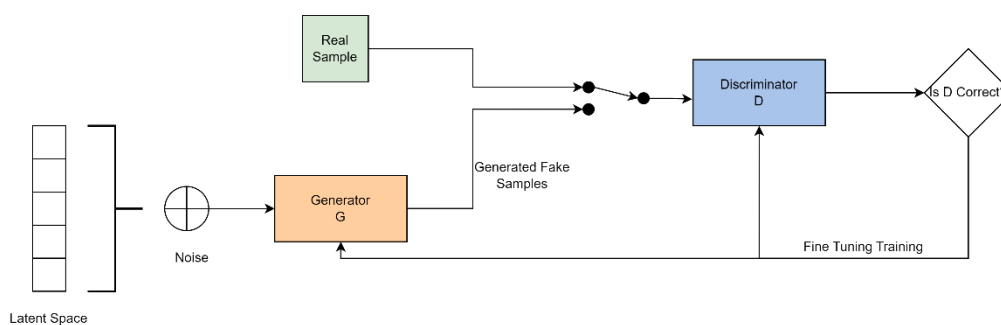


Figure 2.1: Structure of GANs

Figure 2.1 illustrates a fundamental view of the GANs architecture, where a generator creates samples that are evaluated by a discriminator. The generator takes a random seed (input noise) and generates fake data. Where the discriminator is exposed to two types of input data: real data samples from a dataset and fake data produced by the generator. The goal of the generator is to create fake data to fool the Discriminator, which tries to get better at distinguishing real images from synthetically generated ones. The discriminator learns to classify between real and generated fake data, and outcome a judgment of “real or fake” based on its assessment. The bidirectional flow of information and feedback leads to the continuous improvement of both networks. GAN utilize a dual-network architecture where both the generator and discriminator are implemented as Multilayer Perceptrons (MLPs).

2.2.1.1 Multilayer Perceptrons (MLPs)

Multilayer Perceptrons (MLPs) are important neural network architectures that have multiple layers of perceptrons or neurons. These layers are fully connected and each connection has a weight that is adjusted during the training process. (Goodfellow, et al., 2014). The generator's MLP takes a random noise vector as input and transforms it into data that copies the target distribution. The discriminator's MLP then attempts to classify the data as real or fake. The use of MLPs in both networks is critical as it allows for a backpropagation-friendly environment, where gradients can be computed efficiently, enabling the networks to learn and adapt through the adversarial training process.

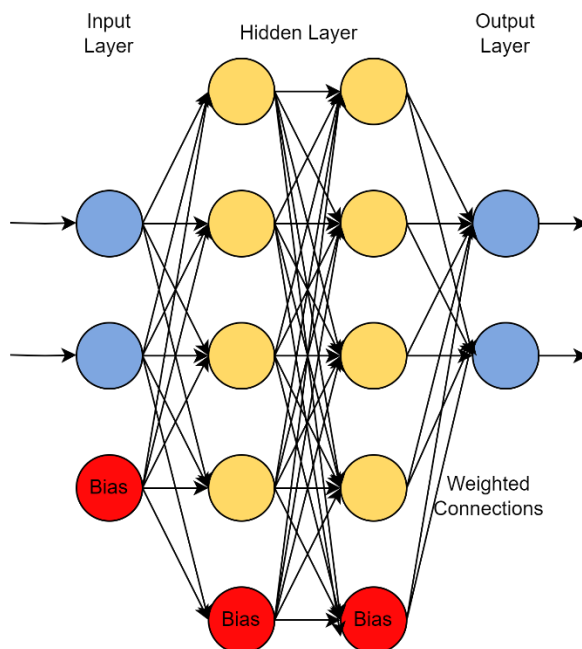


Figure 2.2: Architecture of Multilayer Perceptrons (MLPs)

Figure 2.2 illustrates an architecture of multilayer perceptrons (MLPs) which consists of three layers, which are input layer, hidden layer, and output layer. Each circle represents a neuron and different layers of neurons are interconnected by weighted connections. These weights adjust during the training process. In the first layer, the input layer, the model receives input data. Each of the neurons represents one feature of the input data. Moving on to the next layer, the intermediate layers, process the inputs received from the previous layer by applying weights, biases, and typically non-linear activation functions. These layers extract features and learn representations. The final layer produces the model's output. In a classification task, these neurons typically represent the classes the model is trying to predict. In addition, each layer except for the input layer includes bias neurons. They allow the model to fit the data better. The bias neurons add an extra parameter to the model, which adjusts along with the weights to enhance the model's ability to fit complex patterns. Black lines represent the connections between neurons of sequential layers, and each connections have an associated weight. These weights determine the strength and direction of the influence one neuron has on another.

Besides GAN, MLPs also are adept at the discriminator's role within SeqGANs. It evaluates and provides binary feedback on whether a sequence is

real or generated due to its well-suited structure for classifying fixed-dimensional input data.

2.2.2 Sequence Generative Adversarial Nets

Sequence Generative Adversarial Nets (SeqGANs) have established themselves as a powerful approach for generating realistic and coherent textual sequences (Yu et al., 2016). It integrates reinforcement learning techniques with generative adversarial networks to effectively train a sequence generator. SeqGANs modify the standard of the GAN framework to generate textual data which is sequential and discrete by nature (Yu et al., 2016).

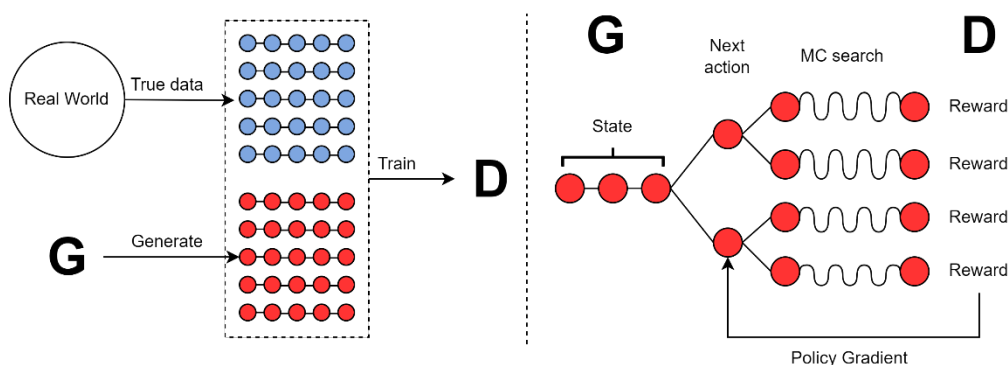


Figure 2.2 demonstrates the structure of SeqGANs. The generator, G operates as a reinforcement learning agent, where each action is the generation of the next token in a sequence. It produces sequences that are evaluated by the discriminator. As for the discriminator, D evaluates the quality of entire sequences, providing rewards to the generator. The generator will then use the rewards to adjust its parameters through policy gradient methods.

Unlike GANs, SeqGAN provides feedback at multiple points in the sequence, allowing the generator to adjust its strategy dynamically, enhancing the overall quality of the sequence generation. Besides that, SeqGANs use Monte Carlo Tree Search (MCTS) to estimate reward signals enabling effective handling of the discrete nature of sequence generation, making it adept at tasks that require maintaining the integrity and contextuality of sequences such as in natural language processing.

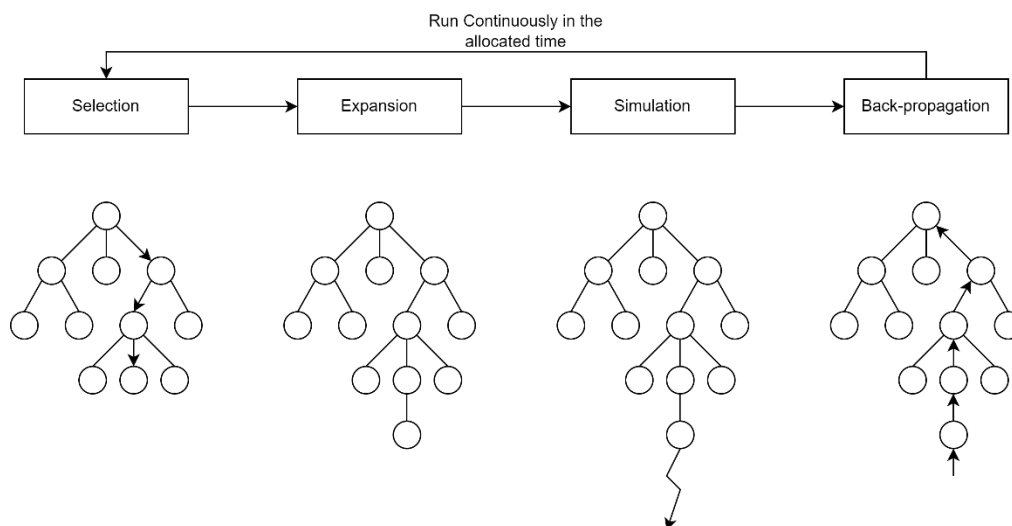


Figure 2.4 presents the Monte Carlo Tree Search (MCTS) process. Monte Carlo Tree Search is a heuristic search algorithm for some kinds of decision processes (Duarte, et al., 2020). MCTS is used to manage the sequence generation process as a decision-making problem, where each choice of a token to add to the sequence can be seen as a move in a game (Duarte, et al., 2020). The algorithm begins with a selection process by traversing the existing nodes of the tree (representing the sequence decisions made so far) and choosing the most promising one based on a policy (Duarte, et al., 2020). Upon reaching a leaf node, the tree is expanded by adding one or more child nodes. This represents the next possible tokens in the sequence. After that, a simulation will be run from the new nodes to the end of the sequence by using a simpler model or random sampling to estimate the outcome (Duarte, et al., 2020). The results of the simulation are then back-propagated through the tree. It will then update the nodes with the new data to better inform future selection processes. SeqGAN incorporates MCTS to evaluate the potential of each decision during sequence generation and the generator to receive intermediate rewards from the discriminator (Duarte, et al., 2020). This assists in modelling the decision-making process involved in sequence generation as a Markov decision process, with the MCTS providing a framework for estimating the long-term rewards of actions (token choices) that do not have immediate feedback (Duarte, et al., 2020).

SeqGANs consist of two key components in their architecture which are Encoder–Decoder Network and adversarial training.

2.2.2.1 Encoder-Decoder Network

The Encoder-Decoder Network is a neural network design that is frequently used for various tasks in machine learning, including natural language processing (NLP) and computer vision. It often employs RNNs like Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU) (Hochreiter & Schmidhuber, 1997; Cho et al., 2014), which allows them to capture the sequential nature of language and generate text that adheres to grammatical rules and stylistic elements (Vaswani et al., 2017). This has led to their successful application in various tasks like machine translation, creative writing, and dialogue systems.

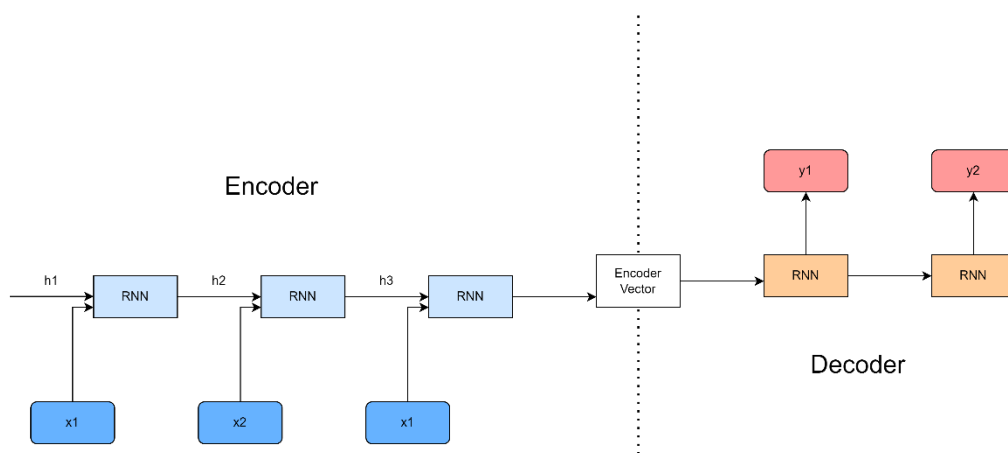


Figure 2.4 illustrates the architecture of an Encoder-Decoder model with Recurrent Neural Networks (RNNs). The encoder gets the essence of the input sequence into a single vector, and the decoder uses this vector to generate the output sequence. The encoder processes an input sequence (x_1, x_2, x_3, \dots) step by step. At each time step, an RNN unit takes an input token and a hidden state from the previous step as input and generates a new hidden state. The encoder vector is the last hidden state (h_3) produced by the encoder. It acts as a representation of the input sequence that obtains the information for the decoder. After that, the decoder will then take the encoder vector as its initial hidden state and start generating the output sequence one step at a time. At each step, it

produces an output token (y_1, y_2, \dots) and a new hidden state that is fed into the next step. In the end, the encoder's final hidden state is used to initialize the decoder's hidden state. It creates a bridge between the encoder and the decoder.

Generative Adversarial Networks (GANs) and Sequence Generative Adversarial Networks (SeqGANs) have provided a striking reflection of the advancements in machine learning architectures tailored to generative tasks. Each of them brings a unique perspective to generative modelling. GANs employ a dual-network structure of Multilayer Perceptrons (MLPs) for generating and discriminating between data, facilitating their success in producing high-quality, continuous data such as images. Their adversarial framework has spurred a multitude of applications, proving the model's versatility and power in various domains.

SeqGANs, on the other hand, extend the GAN framework to the domain of sequence generation. Through the incorporation of Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks in the generator, SeqGANs capture the complexities of sequential data like text. They adeptly handle the intricacies of language, such as grammar and style, necessary for natural language processing applications. Meanwhile, the discriminator, often realized through MLPs or Convolutional Neural Networks (CNNs), provides subtle feedback, enhancing the generator's ability to produce sequences that are consistent and contextually appropriate.

The utilization of Monte Carlo Tree Search (MCTS) within SeqGANs exemplifies the integration of advanced decision-making algorithms with generative models. MCTS in SeqGANs evaluates potential sequence decisions, offering a robust framework for handling the discrete nature of text generation. This integration enables the generator to adapt its strategy dynamically, learning to predict the discriminator's response to different sequences, and refining the sequence generation process.

Table 2.1 and Table 2.2 is the comparison between the two models by highlighting their structures and operational dynamics.

Table 2.1: Feature Comparison of GAN and SeqGAN

Feature	GANs	SeqGANs
Main Idea	<ul style="list-style-type: none"> • Two neural networks compete in a game theory framework. 	<ul style="list-style-type: none"> • Extends GANs to sequence generation with a focus on discrete outputs like text.
Output Type	<ul style="list-style-type: none"> • Continuous data • Example: images 	<ul style="list-style-type: none"> • Discrete sequences • Example: text
Architectural Basis	<ul style="list-style-type: none"> • Multilayer Perceptrons (MLPs) for both generator and discriminator. 	<ul style="list-style-type: none"> • RNNs or LSTMs for the generator to handle sequences • MLPs or CNNs for the discriminator.
Training Feedback	<ul style="list-style-type: none"> • Binary feedback (real vs. fake) at the end of discriminator evaluation. 	<ul style="list-style-type: none"> • Sequential feedback with intermediate rewards using policy gradients • Use to improve learning from partial sequences.
Application Domain	<ul style="list-style-type: none"> • Image generation • Art creation • Photo enhancement 	<ul style="list-style-type: none"> • Text generation • Machine translation • Dialogue system
Key Innovations	<ul style="list-style-type: none"> • Eliminates the need for Markov chains • Inference during learning <ul style="list-style-type: none"> • Incorporate complex, sharp distributions. 	<ul style="list-style-type: none"> • Integrates Monte Carlo Tree Search to evaluate policy rewards and manages the discrete nature of the text.
Usability and Flexibility	<ul style="list-style-type: none"> • High usability in visual contexts • Less effective with discrete data. 	<ul style="list-style-type: none"> • High flexibility with sequential data • Models dependencies and contexts effectively.

Table 2.2: Training Process Comparison of GAN and SeqGAN

Training Aspect	GANs	SeqGANs
Objective	<ul style="list-style-type: none"> • Discriminator maximizes real vs. fake classification accuracy 	<ul style="list-style-type: none"> • Generator maximizes expected reward through discriminator's evaluation

	<ul style="list-style-type: none"> • Generator minimizes discrimination 	
Complexity Handling	<ul style="list-style-type: none"> • Manages pixel-level data distributions 	<ul style="list-style-type: none"> • Manages temporal dependencies and contextual relevance
Feedback Mechanism	<ul style="list-style-type: none"> • At the end of the discriminator's evaluation 	<ul style="list-style-type: none"> • Throughout the sequence generation
Optimization Technique	<ul style="list-style-type: none"> • Backpropagation • Adversarial training 	<ul style="list-style-type: none"> • Backpropagation • Policy gradient methods • Monte Carlo Tree Search (MCTS)
Typical Use Case	<ul style="list-style-type: none"> • Static data generation 	<ul style="list-style-type: none"> • Dynamic, contextual data generation

The integration of advanced neural network architectures and strategic decision-making algorithms in GANs and SeqGANs has significantly advanced the ability of generative models. SeqGANs have opened up new possibilities in the generation of text. It shows the adaptability and transformative potential of the models across various domains of application.

2.3 Comparative Analysis of Policy Gradient Methods

Policy gradient acts as an agent that interacts within the environment and offers a framework to train the model in reinforcement learning. The SeqGAN model helps on tasks where decision-making is sequential and the objectives are long-term. The policy gradient's goal is to optimize the policy directly (a model's strategy for action selection) by maximizing the expected cumulative reward. This optimization is achieved by adjusting the policy parameters in a direction to make the probability of successful actions increases (Sutton & Barto, 2018).

2.3.1 Policy Gradient Algorithms

Several policy gradient methods have been proposed and each of them has its approach to balance the two critical aspects of learning in uncertain environments: exploration and exploitation. This section will describe REINFORCE (Williams, 1992), which serves as the basis for many subsequent algorithms; Trust Region Policy Optimization (TRPO), known for its stable convergence properties (Schulman, et al., 2015) and Proximal Policy

Optimization (PPO), which simplifies and improves upon TRPO (Schulman, et al., 2017).

2.3.1.1 REINFORCE

REINFORCE is one of the policy gradient algorithms fundamental to reinforcement learning. It was introduced by Williams in 1992. It stands out for its straightforward approach to policy optimization and utilizes Monte Carlo methods to estimate the gradient of the expected reward directly. In contrast to value-based methods, its simplicity allows the direct optimization of the policy without the need for a value function estimator (Williams, 1992). The core principle of REINFORCE lies in the utilization of the complete returns from episodes to perform policy updates. It alters policy parameters in a way that raises the likelihood of actions that lead to higher returns. This is accomplished by computing the gradient of the expected return to the policy parameters, which are then used to execute gradient ascent. The predicted return is calculated through sampling, and the update is proportionate to the return. The gradient of the log probability of the taken actions will integrate exploration naturally into the policy updates. Its core equation can be stated as

$$\Delta\theta = \alpha \cdot \nabla\theta \log\pi\theta(s, a) \cdot R \quad (2.1)$$

Where $\Delta\theta$ represents the change in the policy parameters, α is the learning rate, $\nabla\theta \log\pi\theta(s, a)$ is the gradient of the logarithm of the policy $\pi\theta$ for its parameters θ , given that s is the state and a is the action. R is the reward signal, which assesses the quality of action, a taken in the state, s . The reward R can be particularly challenging to define for text generation, as it often only becomes clear at the end of the sequence.

The integration of REINFORCE within SeqGAN is a natural progression, given SeqGAN's structure that mirrors an environment with sequential decision-making and delayed rewards, typical in natural language processing tasks. SeqGAN modifies the traditional GAN framework to tackle the discrete and sequential nature of text, rendering standard backpropagation

methods ineffective due to the non-differentiability of the sampling process. REINFORCE comes into play as an elegant solution to this problem. By formulating text generation as a reinforcement learning task, REINFORCE allows SeqGAN to navigate the sequential construction of text where the signal of success—a cohesive and contextually appropriate sequence—is only apparent at the end (Yu, et al., 2017).

Algorithm 2: REINFORCE

```

01: Input: a differentiable policy parameterization  $\pi(a|s, \theta)$ 
02: Algorithm policy parameter  $\theta \in \mathbb{R}^{d'}$  (example, to 0)
03: for each episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_{-1}$ , following  $\pi(\cdot | \cdot, \theta)$ 
04:     for  $t = 1$  to  $T - 1$ , do
           
$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

           
$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t, \theta)$$

05:     end for
06: end for
07: return  $\theta$ 

```

Algorithm 2.1: Algorithm of REINFORCE

Chen et al. (2018) demonstrated the efficacy of REINFORCE in a complex, real-world recommender system, suggesting its potential when adapted to SeqGAN for text generation. The critical observation from their implementation was the need for off-policy correction to mitigate biases from historical data. This insight translates into SeqGAN's learning environment, where the generator must adjust based on the discriminator's evolving criteria for what constitutes 'real' text, and hence the policy needs to be robust to shifts in the data distribution.

While REINFORCE provides a direct method for optimizing policies based on long-term rewards, its application within SeqGAN is not without challenges. The high variance in gradient estimates inherent to REINFORCE can lead to unstable training and slow convergence, which is particularly

problematic in the context of SeqGAN where the policy space is vast and complex. Methods to reduce variance, such as introducing a baseline or employing reward-shaping techniques, are essential considerations for improving SeqGAN's training efficiency.

2.3.1.2 Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) is a policy gradient method that tackles the stability and efficiency difficulties raised by previous policy gradient methods such as REINFORCE. Originally proposed by Schulman et al. (2015), TRPO aims to take the largest possible improvement step on a policy without causing the collapse of performance. It aims to make it highly suitable for problems involving high-dimensional, continuous action spaces (Schulman et al., 2015).

TRPO extends the standard policy gradient approach by incorporating a trust zone constraint to limit the extent of policy changes. This constraint is implemented using the KL-divergence to ensure the new policy is not too far from the old policy, thus maintaining the stability of the learning updates. The objective function of TRPO can be expressed as

$$\max_{\theta} E_{s,a} \sim \pi_{old} \left[\frac{\pi_{\theta}(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s, a) \right] \quad (2.2)$$

subject to

$$E_s \sim \pi_{old} [D_{KL}(\pi_{old}(\cdot|s) \parallel \pi_{\theta}(\cdot|s))] \leq \delta \quad (2.3)$$

Where π_{θ} and π_{old} are the new and old policies parameterized by θ , $A^{\pi_{old}}$ is the beneficial function under the old policy and δ is a small constant that defines the extent of the trust region. The figure below presents the Algorithm of TRPO.

Algorithm 3: Trust Region Policy Optimization

Require: Hyperparameters: Maximum number of backtracking steps K ;
Backtracking coefficient α ; KL-divergence limit δ .

- 01: Input: initial parameters θ_0 , initial value function ϕ_0 .
- 02: **for** $k = 0, 1, 2, 3 \dots$ **do**
- 03: Collect a set of trajectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 04: Calculate rewards-to-go \hat{R}_t .
- 05: Calculate advantage estimates, \hat{A}_t (using any method of benefit estimation), based on the current value function. V_{ϕ_k}
- 06: Estimate gradient policy as

$$\text{the } \hat{g}_k = \frac{1}{|D_k|} \sum_{\tau \in D_k} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) |_{\theta_k} \hat{A}_t \cdot$$

- 07: Calculate using the conjugate gradient algorithm.

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k$$

Where the Hessian of the sample average KL-divergence., denoted as \hat{H}_k .

- 08: Retrace the line of search to update the policy.

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k$$

where $j \in \{0, 1, 2, \dots, K\}$ is the lowest number that improves sample loss and meets the sample KL-divergence requirement.

- 09: Fit the value function using regression on mean-squared error:

$$\text{the } \phi_{k+1} = \arg \min \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2$$

- 10: typically via some gradient descent algorithm.

- 11: **end for**

Algorithm 2.2: Algorithm of TRPO

Implementing TRPO in the SeqGAN framework helps to optimize the generator's policy in generating sequences. Given the sequential and discrete nature of text generation tasks in SeqGAN, TRPO's strength in parameter updates plays a crucial role in maintaining the stability of training when the

discriminator's feedback changes the landscape of the policy's performance surface. The stability introduced by the trust region helps mitigate issues related to catastrophic forgetting and sharp performance degradation.

Despite its advantages, the incorporation of TRPO into SeqGAN presents unique challenges which discrete action spaces. TRPO is inherently designed for continuous actions that complicate its direct application to the discrete token selections in SeqGAN. Adaptations are included when modifying TRPO to support discrete actions that approximate the continuous methods. This method will introduce biases or inefficiencies (Schulman et al., 2015). Besides that, the other challenge is the computational complexity of TRPO. The use of second-order optimization methods (i.e., calculating the Hessian) is computationally expensive which causes TRPO to become less scalable for large sequence models compared to first-order methods like those used in Proximal Policy Optimization (PPO) (Schulman, et al., 2015).

TRPO offers a theoretically sound and stable approach for optimizing policy gradients in challenging environments. While SeqGAN's application is non-trivial, it provides a framework for improving the strength and reliability of sequence generation models in the face of dynamic and complex discriminator behaviours (Schulman, et al., 2015).

2.3.1.3 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a policy gradient method that refines the ideas of Trust Region Policy Optimization (TRPO) into a more practical framework. PPO was introduced by Schulman et al. in 2017. It retains the core concept of trust regions but simplifies the optimization process by using a clipped surrogate objective. This has made it computationally less intensive and easier to implement (Schulman et al., 2017). PPO addresses the complexities of policy optimization by clipping the probability ratio which discourages large deviations from the old policy. The clipped objective function of PPO is defined as

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

where θ is the policy parameter, \hat{E}_t denoted the empirical expectation over timesteps. r_t is the ratio of the probability under old and new policy. \hat{A}_t is the estimated advantage time t . ϵ represent the hyperparameter which is usually around 0.1 – 0.2.

Algorithm 4: Proximal Policy Optimization (PPO)

```

01: Initialize  $\mu: s \rightarrow R^{m+1}$  and  $\sigma: S \rightarrow \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_{m+1})$ 
02: for  $I = 1$  to  $M$  do
03: Run policy  $\pi_{\theta} \sim N(\mu(s), \sigma(s))$  fit  $T$  timesteps and collect
       $(s_t, a_t, r_t)$ 
04:       for  $t = 1$  to  $T - 1$ , do
              
$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

              
$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t, \theta)$$

05:       end for
06: end for
07: return  $\theta$ 

```

Algorithm 2.3: : Algorithm of PPO

PPO can compute advantage estimates and optimize the clipped surrogate objective through stochastic gradient ascent. Furthermore, PPO allows for multiple epochs of minibatch updates per data sample collected. It is contrasted with the single update per sample approach seen in other policy gradient methods.

When PPO is applied to SeqGAN, it will optimize the generator's policy for sequence generation tasks, such as text. The PPO algorithm accommodates the unique challenges posed by the sequential and discrete nature of text generation, where traditional policy gradient methods like REINFORCE can struggle due to high variance in gradient estimates. The clipping mechanism in PPO aims to prevent excessively large updates. As the generator incrementally constructs a sequence, it will receive delayed and potentially sparse rewards (Schulman, et al., 2017).

Adapting PPO, typically used in continuous spaces, to SeqGAN's discrete token space often involves using softmax policy representations or other modifications to the PPO. This adaptation is crucial to accommodate the discrete probability distributions inherent in SeqGAN (Schulman, et al., 2015). Besides that, PPO incorporates techniques such as Generalized Advantage Estimation (GAE) to reduce variance and improve the stability of training, which is especially beneficial for the complex optimization landscape of SeqGAN (Schulman, et al., 2015).

PPO represents an advancement in policy gradient algorithms with the ability to maintain robust performance while being computationally more efficient than its predecessor, TRPO. Its application to SeqGANs offers an approach to generating high-quality sequences by leveraging its stability and efficiency in policy updates. The PPO algorithm's potential to scale to complex models and tasks positions it as a preferred choice for SeqGANs and other sequence generation models in reinforcement learning applications (Schulman, et al., 2017)

Table 2.3: Comparison of Policy Gradient Algorithms

Feature	REINFORCE	TRPO	PPO
Gradient Estimation	<ul style="list-style-type: none"> • High variance • Simple estimate 	<ul style="list-style-type: none"> • Low variance • Uses trust regions 	<ul style="list-style-type: none"> • Clipped objective to manage variance
Sample Efficiency	<ul style="list-style-type: none"> • Low 	<ul style="list-style-type: none"> • Moderate 	<ul style="list-style-type: none"> • High
Stability and Strength	<ul style="list-style-type: none"> • Less stable 	<ul style="list-style-type: none"> • More stable due to trust regions 	<ul style="list-style-type: none"> • Balances between stability and sample efficiency
Complexity and Implementation	<ul style="list-style-type: none"> • Simple 	<ul style="list-style-type: none"> • Complex 	<ul style="list-style-type: none"> • Less complex than TRPO, easier to implement

Common Use Cases	<ul style="list-style-type: none"> • Small-scale problems 	<ul style="list-style-type: none"> • High-dimensional control tasks 	<ul style="list-style-type: none"> • Broad range of applications including robotics and NLP
------------------	--	--	--

Table 2.3 offers an overview comparison of the attributes of each policy gradient method. The complexity and implementation row reflects the relative ease or difficulty of coding and executing each algorithm.

Table 2.4: Sustainability for SeqGAN and Test Generation

Feature	REINFORCE	TRPO	PPO
Exploitation Trade-off	<ul style="list-style-type: none"> • High exploration 	<ul style="list-style-type: none"> • Balance • Uses KL divergence to limit policy updates 	<ul style="list-style-type: none"> • Adaptive • Strike a balance with clipped objectives
Learning from Sparse Rewards	<ul style="list-style-type: none"> • Struggles • High variance 	<ul style="list-style-type: none"> • Better • Trust regions prevent drastic policy updates 	<ul style="list-style-type: none"> • Good • Uses multiple epochs to learn from limited data
Training Overhead	<ul style="list-style-type: none"> • Minimal 	<ul style="list-style-type: none"> • Significant due to second-order methods 	<ul style="list-style-type: none"> • Moderate • Leverages first-order methods
Performance in Text Generation Tasks	<ul style="list-style-type: none"> • Varies, requires more iterations 	<ul style="list-style-type: none"> • Robust but computationally intensive 	<ul style="list-style-type: none"> • Superior in balancing speed and quality

Table 2.4 highlights the way each algorithm might perform given the unique challenges posed by the sequential and discrete nature of text. The training overhead and response to non-stationarity are particularly pertinent for SeqGANs, as the generative model continuously evolves during training, requiring the algorithm to adapt to the shifting data distribution efficiently.

2.4 Architecture Innovation

The architecture of Generative Adversarial Networks has seen significant evolution since the beginning. The push for innovation has been the diverse and

growing range of applications, each posing unique challenges that demand specialized solutions. In text generation, SeqGAN has stood out by incorporating these architectural innovations, each adaptation serving to improve the model's performance in generating coherent and contextually rich textual content.

2.4.1 Long short-term memory (LSTM)

Long short-term memory (LSTM) units (Hochreiter & Schmidhuber, 1997) are specifically designed to overcome the loss of gradient problem that troubles standard recurrent neural networks (RNNs). LSTMs use a sequence of gates, known as input, forget, and output gates, to manage the flow of information. These gates collectively decide which data should be retained or discarded, thus maintaining a stable gradient across learning sequences. The input gate regulates the amount of new information that enters the cell state, the forget gate controls the data that is erased from the cell state, and the output gate determines the next hidden state. This complex gating mechanism allows LSTMs to preserve information over extended periods and improves their capability to model sequences with complex structures (Hochreiter & Schmidhuber, 1997).

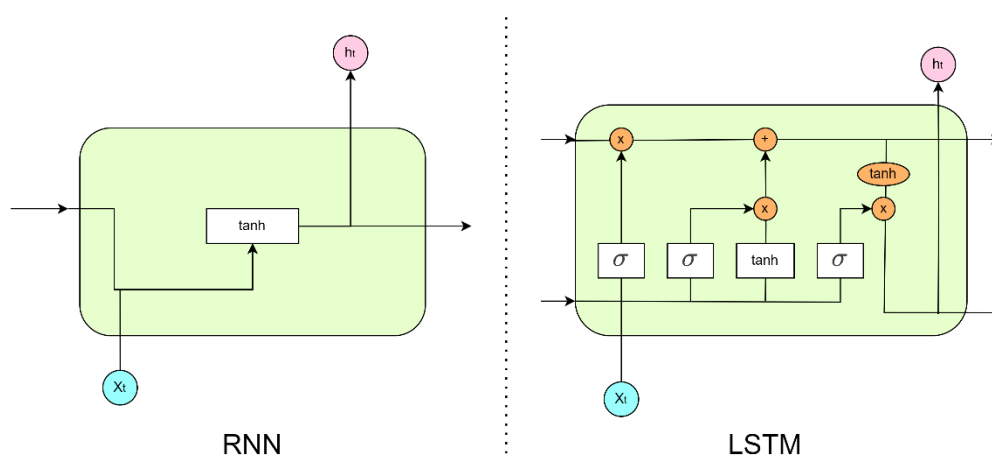


Figure 2.6: Unit of RNN and LSTM

Figure 2.9 illustrates the structure of RNN and LSTM where both are designed to handle sequences of data x_t where t is the time steps, but each of the units manages information differently. Where input x_t is the current input

of the sequence, \tanh is the hyperbolic tangent activation function that generates a new state from the input and previous hidden state, and hidden state h_t represents the output state that captures information from the current input and previous inputs over time, t . As shown at the left of the figure, the RNN unit takes the current input x_t and processes it through a single layer with a \tanh activation function to produce the hidden state h_t . This hidden state is then fed back into the RNN unit at the next time step, along with the next input in the sequence.

On the other side of the picture, the LSTM unit's internal memory is shown, which carries information over time steps. Gates determine how information is added or deleted from the cell state. LSTM is made up of three gates: forget gates (σ) that discard information from the cell state, input gates (σ and \tanh) that add new information to the cell state, and output gates (σ) that determine which information from the cell state is used to generate the output hidden state. Each gate in the LSTM unit uses a combination of the sigmoid (σ) and \tanh activation functions. The sigmoid function returns a value ranging from 0 to 1, which is used to scale the contribution of other operations. The \tanh function regulates the nonlinear transformation of the data, scaling the values to be between -1 and 1. In an LSTM unit, the operations are more complex due to the multiple gates that manage memory and output. The forget gate selects which bits of the cell state to keep or erase. The input gate determines which values from the cell state are updated and what new values are added. Finally, the output gate determines which part of the cell state will be used to generate the output h_t .

Knowing the ability of LSTM that remember information for a long period, it became ideal for applications involving sequential data such as text, speech, and music. There are various applications of LSTMS across different fields such as text generation, speech recognition, and music generation.

Text Generation

LSTM can generate coherent paragraphs of text, capture long-range dependencies, and manage multiple themes at once. A study by Sutskever et al.

(2014) demonstrated the ability of LSTMs to perform sequence-to-sequence learning. They were employed to generate text at both character and word levels efficiently. This ability is not only impressive in terms of the linguistic quality of the generated text but also in the variety of applications it enables, from automated story generation to interactive chatbots.

Speech Recognition

LSTMs are used to convert audio clips containing spoken language into text by understanding the temporal dependencies in spoken language in speech recognition. Graves et al. (2013) utilized LSTMs to develop a speech recognition system that operates directly on the spectrogram of spoken audio. This illustrated the network's ability to handle raw audio data and perform end-to-end speech recognition. This technology not only powers popular virtual assistants but is also crucial in accessibility technologies for those with speech impairments.

Music Generation

The application of LSTMs in music generation demonstrates their versatility. Eck and Schmidhuber (2002) were pioneers in using LSTMs for generating blues music. They had demonstrated that these networks could learn not just the notes but also the timing and style of blues music from raw audio. LSTMs help generate new music pieces that mimic the style of a given training dataset.

2.4.2 Gated Recurrent Unit (GRU)

Cho et al. (2014) introduced GRU Units, which simplify the LSTM architecture by combining input and forget gates into a single "update gate" and integrating the cell state with the hidden state. GRUs maintain the efficiency of LSTMs but with fewer parameters, resulting in faster computations and simpler models. The update gate in GRUs assists the model in determining how much past information (from earlier time steps) to pass along, whereas the reset gate allows the model to decide how much past information to discard. These features make GRUs particularly useful for SeqGAN models where computational efficiency and model simplicity are desired (Cho et al., 2014).

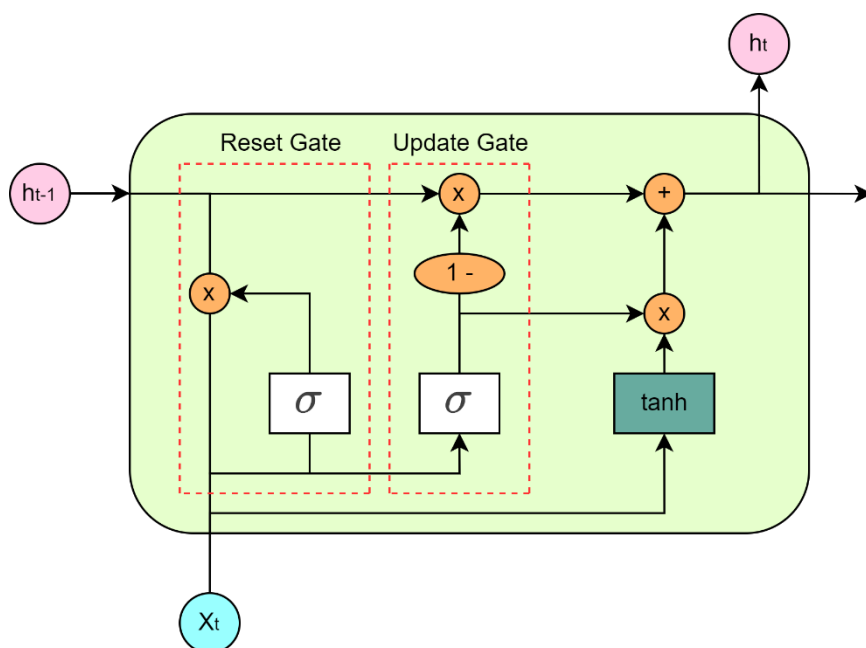


Figure 2.7: Unit of GRU

Figure 2.7 displays the internal structure of a GRU. GRUs are comparable to LSTMs but have a simpler structure. They are intended to tackle the vanishing gradient problem of traditional RNNs. x_t represent the new input at time step t , whereas the h_{t-1} represent the hidden state from the previous time step $t - 1$. The reset gate controls how much of the prior information (hidden state) must be forgotten. It uses the current input x_t and the previous hidden state h_{t-1} to calculate a reset factor using a sigmoid activation (σ), which ranges between 0 and 1. After that, the update gate determines how much of the previous information (previous hidden state) will transfer over to the current hidden state. It also calculates an update factor using a sigmoid activation, based on the current input x_t and the previous hidden state h_{t-1} . \tilde{h}_t known as the candidate's hidden state. It is a mixture of the current input and the previous hidden state, modulated by the reset gate. It uses a tanh activation to keep the values between -1 and 1. The new hidden state, h_t is a mixture of the old hidden state and the candidate's hidden state. The update gate determines how much of the candidate's hidden state is being used to update the current hidden state.

In a nutshell, the GRU contains two gates: the reset gate which determines how to integrate the incoming input with the prior memory, and the update gate determines how much of the previous memory to retain. If the reset

gate is near to zero, the hidden state is compelled to ignore the previous hidden state and reset using only the current input. This essentially allows the GRU to discard information that is no longer useful for future steps, which aids in the learning of long-term dependencies. The update gate assists the model in determining how much of the previous knowledge should be passed on to the future.

2.4.3 Attention Mechanisms

Attention mechanisms in neural networks have revolutionized the way models handle and interpret data. Attention can be described as a function that maps a query and a set of key-value pairs to an output. They selectively focus on areas of the input that are regarded most relevant to the job at hand, enhancing the model's capacity to execute tasks such as language translation, image recognition, and sequence prediction. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Given a query q , keys k_1, \dots, k_n , and values v_1, \dots, v_n , the attention function can be expressed as:

$$\text{Attention}(q, K, V) = \sum_i \alpha_i v_i \quad (2.4)$$

where $\alpha_i = \text{softmax}(\text{score}(q, k_i))$

When applied to SeqGANs as Attention SeqGAN, it significantly enhances the model's context awareness. By focusing on relevant parts of the input sequence, the model maintains contextual coherence over extended sequences, a critical aspect in text generation (Vaswani et al., 2017).

SeqGANs have extended the use of generative adversarial networks to sequence generation, but the integration of attention mechanisms has brought about a notable improvement in their performance. Attention in SeqGANs facilitates a more focused generation process by aligning generated sequences to become closer to contextual relevance and coherence.

2.4.4 Conditional GANs

Conditional GANs (cGANs) represented a leap forward in GAN architecture by integrating additional information to guide the generation process (Goodfellow, et al., 2014). This advancement allowed for targeted generation, where the model could be conditioned on labels or types of data, enabling more control over the output. Applied to SeqGAN, such as in Conditional SeqGAN, this innovation has profoundly impacted text generation. It permits the thematic elements of the generated text to be directed, resulting in content that can be tailored to specific topics or styles, enhancing both relevance and diversity in generated narratives (Guo et al., 2021).

Table 2.5: Comparisons of SeqGAN Architectures Innovation

Feature	Basic SeqGAN Model	GRU	Attention Mechanism	Conditional SeqGAN
Architecture Type	<ul style="list-style-type: none"> Recurrent Neural Network (RNN) 	<ul style="list-style-type: none"> Deep Recurrent Neural Networks (Deep RNN) 	<ul style="list-style-type: none"> RNN with Attention Layers 	<ul style="list-style-type: none"> RNN with Conditional Inputs
Key Components	<ul style="list-style-type: none"> Simple RNN layers 	<ul style="list-style-type: none"> GRU layers for handling long-term dependencies 	<ul style="list-style-type: none"> Attention layers will focus on relevant parts of the input sequence 	<ul style="list-style-type: none"> Additional inputs (tags or labels) condition the generation process
Advantages	<ul style="list-style-type: none"> Simplicity and efficiency Quick to train 	<ul style="list-style-type: none"> Better at capturing long-range dependencies Improves the quality of generated sequences 	<ul style="list-style-type: none"> Increases the model's focus and relevance Enhances coherence of the generated text 	<ul style="list-style-type: none"> Generates context-specific sequences Increases utility and applicability of the model
Limitations	<ul style="list-style-type: none"> Struggles with long sequences Prone to vanishing gradient problem 	<ul style="list-style-type: none"> Computationally more intensive Requires more data to train effectively 	<ul style="list-style-type: none"> Can be complex to implement Requires careful tuning of attention mechanisms 	<ul style="list-style-type: none"> More complex training process Needs well-annotated data for conditioning
Typical Applications	<ul style="list-style-type: none"> Short text generation Quick prototyping of text generators 	<ul style="list-style-type: none"> Content creation Tasks requiring an understanding of context over longer sequences 	<ul style="list-style-type: none"> Detailed content generation Example: medical or legal documents 	<ul style="list-style-type: none"> Style-specific writing Example: poetic forms, technical manuals

Table 2.5 outlines the comparisons of various SeqGAN architecture innovations. The evolution from basic RNN structures to more sophisticated configurations like GRU layers and attention mechanisms signifies a substantial advancement in handling the subtlety demands of sequence generation tasks. These innovations not only enhance the quality and coherence of the generated texts but also expand the model's applicability to a broader range of text-generation circumstances. (Hochreiter & Schmidhuber, 1997; Vaswani, et al., 2017; Yu, et al., 2017)

Furthermore, the introduction of conditional architectures in SeqGANs allows for the generation of context-specific texts, which is a significant step forward in the customization and relevance of the outputs produced. This capability is particularly valuable in fields that require high levels of precision and adaptability in text generation, such as creative writing and technical documentation (Yu et al., 2017).

Progressive training approaches, exemplified by Progressive GANs (ProGANs), incrementally increase the complexity of the model during training (Karras et al., 2018). This approach applied to SeqGAN as Progressive SeqGAN, can facilitate a structured approach to complexity in text generation. Starting with simpler structures and gradually advancing to complex narratives can lead to more coherent and thematically consistent content generation (Shi et al., 2018).

Table 2.6: Comparison of Training Techniques for SeqGAN

Training Technique	Standard Training (RL Only)	Hybrid Training (Supervised + RL)	PPO
Key Features	<ul style="list-style-type: none"> • Policy Gradient • REINFORCE Algorithm 	<ul style="list-style-type: none"> • Supervised Pre-training • Reinforcement Learning Fine-tuning 	<ul style="list-style-type: none"> • Clipped Surrogate Objective • Adaptive KL Penalty
Advantages	<ul style="list-style-type: none"> • Directly optimizes for the final objective • Simpler implementation with fewer hyperparameters 	<ul style="list-style-type: none"> • Faster convergence initially • Supervised pre-training can lead to a more stable RL phase • Mitigates traditional RL instabilities 	<ul style="list-style-type: none"> • Reduces the risk of destructive large policy updates • Maintains efficient exploration-exploitation balance • Higher stability and better performance
Limitations	<ul style="list-style-type: none"> • High variance in policy updates • Can be sample inefficient • Struggles with stability and often requires careful tuning 	<ul style="list-style-type: none"> • Requires an accurately labelled dataset for supervised training • More complex setup and potentially higher computational overhead 	<ul style="list-style-type: none"> • Computationally more demanding • Needs meticulous tuning of the clipping parameter and adaptation rate • Potentially complex integration with SeqGAN architecture
Performance Indicators	<ul style="list-style-type: none"> • Speed of initial learning • Quality and diversity of generated text 	<ul style="list-style-type: none"> • Quality of pre-trained model (supervised phase) • Improvement in text generation (RL phase) 	<ul style="list-style-type: none"> • Overall stability and quality of training • Consistency and diversity of the generated text
Suitability for Long Text Generation	<ul style="list-style-type: none"> • Low to moderate (depending on the specifics of the policy gradient implementation) 	<ul style="list-style-type: none"> • Moderate to high (effective pre-training can significantly enhance abilities) 	<ul style="list-style-type: none"> • High (optimized policy updates lead to better long-term performance)
Typical Applications	<ul style="list-style-type: none"> • Basic text generators • Prototyping new model architectures 	<ul style="list-style-type: none"> • Developing models for complex narrative generation • Models requiring stable foundations before fine-tuning 	<ul style="list-style-type: none"> • Advanced narrative generation • High-quality text generation tasks requiring subtlety control over style and coherence

Table 2.6 highlights the impact of specific architectural innovations on GANs and SeqGANs, with a focus on the comparative analysis of these impacts in the context of text generation.

The integration of these architectural innovations into SeqGANs has been important in advancing text generation abilities. By leveraging advances from traditional GAN architectures, SeqGANs have become more adept at handling the subtleties and complexities of language. Whether through enhanced thematic control, better context maintenance, or the generation of stylistically diverse content, these innovations have broadened the potential of SeqGANs beyond short sequences to more complex forms of narrative (Guo et al., 2021; Vaswani et al., 2017; Chen et al., 2016; Karras et al., 2018).

2.5 Evaluation Metric

Evaluation metrics are quantitative indicators that analyze a statistical or machine learning model's performance and effectiveness, such as proximity to the target distribution, diversity, and coherence. These measures provide a quantitative basis for measuring the performance of generative models, allowing objective comparisons between different models or techniques.

When analyzing a machine learning model, it is vital to consider its predictive abilities, generalizability, and overall quality. Evaluation metrics provide objective standards to evaluate these qualities. The evaluation metrics used depend on the problem domain, data type, and desired outcome.

Bilingual Evaluation Understudy (BLEU)

The Bilingual Evaluation Understudy (BLEU) score, introduced by Papineni et al. (2002), is one of the earliest metrics adopted for evaluating machine-translated text against a set of reference translations. Despite its origin in translation, it's applied to any text generation task to measure the overlap of n-grams between the generated text and reference text, thus gauging syntactic consistency. However, BLEU's limitations are notable; it does not account for semantic coherence and can score highly on texts that are nonsensical to human readers.

Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

ROUGE is a set of metrics designed to evaluate the quality of summary texts proposed by Lin (2004). It examines the overlap of n-grams, word sequences, and word pairings between the generated and reference texts. ROUGE is known for its emphasis on recall, making it especially appropriate for jobs like summarization where capturing content from the source is critical.

Metric for Evaluation of Translation with Explicit ORDERing (METEOR)

Metric for Evaluation of Translation with Explicit Ordering (METEOR) is introduced by Banerjee and Lavie (2005), and extends beyond n-gram matching to include synonymy and paraphrase matching. Its goal is to address some BLEU score shortcomings. It attempts to align more closely with human judgment by considering the variety of ways in which ideas can be expressed linguistically, and it's praised for its balance between precision and recall.

Perplexity

Perplexity is a metric used to evaluate language models, reflecting how well a probability distribution predicts a sample. It's a measure of the model's uncertainty, with lower values indicating better predictive performance. While perplexity provides insight into the model's fluency, it doesn't directly measure how coherent or contextually appropriate the generated text is.

Negative log-likelihood (NLL)

Negative log-likelihood (NLL) is often used as a loss function during the training of language models, including SeqGANs. It measures how well the model predicts a sequence. In the context of evaluation, lower NLL values indicate that the model assigns higher probabilities to the real data, signifying better performance. However, NLL may not always correlate with human judgments of quality, as models with lower NLL can still generate nonsensical outputs.

Human Evaluation

Human evaluation is regarded as the gold standard for determining the quality, coherence, and relevance of produced text, even if Oracle NLL and BLEU provide quantitative metrics. It is dependent on the subjective evaluation of human raters; higher ratings correspond to higher caliber-produced content. Although human review is a costly, time-consuming, and intrinsically subjective process, it is capable of capturing subtleties that automated measurements could overlook.

Table 2.7 below highlights three commonly used evaluation metrics: BLUE, ROUGE, METEOR, Perplexity, NLL and human evaluation. Each metric is tailored for specific tasks and scenarios, with its unique strengths and weaknesses.

Table 2.7: Comparison of various Evaluation Metrics

Metric	Description	Advantages	Disadvantage
BLEU	<ul style="list-style-type: none"> Indicates the geometric mean of the modified n-gram precision. 	<ul style="list-style-type: none"> High precision in evaluation, widely used in machine translation 	<ul style="list-style-type: none"> Not correlate well with human judgment, especially with higher-order n-grams
ROUGE	<ul style="list-style-type: none"> Compares overlapping n-grams between the generated text and reference texts 	<ul style="list-style-type: none"> Good for evaluating summarization Focuses on recall 	<ul style="list-style-type: none"> Less emphasis on lexical choice precision can be gamed with generic responses
METEOR	<ul style="list-style-type: none"> Harmonizes precision and recall of unigrams between generated and reference texts 	<ul style="list-style-type: none"> Higher correlation with human judgment than BLEU Considers synonyms and stemming 	<ul style="list-style-type: none"> Computationally intensive, more complex to implement
Perplexity	<ul style="list-style-type: none"> Evaluates the likelihood of the sequence given the model 	<ul style="list-style-type: none"> Lower perplexity indicates better performance; straightforward to calculate 	<ul style="list-style-type: none"> Does not account for grammatical correctness or relevance
NLL	<ul style="list-style-type: none"> Measures the model's prediction error. 	<ul style="list-style-type: none"> Directly related to the model's objective function during training Able to provide a clear indication of how well the model has learned the data distribution. 	<ul style="list-style-type: none"> Does not reflect the quality of generated sequences in a way that correlates with human judgment.
Human Evaluation	<ul style="list-style-type: none"> Evaluate Overall text quality 	<ul style="list-style-type: none"> Considered as the gold standard Captures nuances in text quality 	<ul style="list-style-type: none"> Time-consuming and costly Subject to evaluator bias

2.6 Regulatory Mechanisms and Penalties

The penalty is a regularization mechanism used to modify the reward signal during the training process, influencing the behaviour of the generator network. These penalties are crucial in regularizing the loss function to promote desirable outputs by the generator network (Yu, et al., 2017).

In SeqGAN, penalties are deployed as additional terms incorporated into the loss function to include the reward signal received by the generator (Yu et al., 2017). This reward signal, fundamental to policy gradient methods employed in SeqGAN, encapsulates the evaluation of generated sequences against a reward model or discriminator (Yu et al., 2017). The primary objective of penalties is to guide the generator toward producing outputs that not only maximize the likelihood of generating real data but also adhere to specific constraints or exhibit desired properties (Li et al., 2017). By augmenting the reward signal, penalties play a crucial role in encouraging the generator to explore the output space more effectively, thus facilitating the generation of high-quality sequences that are better suited to the task's goal (Li et al., 2017).

However, during the training process, the generator may exploit certain flaws or weaknesses in the discriminator and lead to mode collapse or other undesirable behaviours. To mitigate these issues, various penalty terms can be added to the objective function of the generator and/or discriminator networks. These penalty terms act as regularizers to encourage the networks to learn more robust and diverse representations (Martin Arjovsky, et al., 2017)). One of the key functions of penalties in SeqGAN is to promote diversity within the generated sequences (Zhang, et al., 2017). Diversity is important to ensure that the generator produces a wide range of outputs that capture the inherent variability present in the data distribution (Zhang, et al., 2017). Penalties designed to encourage diversity may penalize the generator for producing repetitive or similar sequences, thereby incentivizing it to explore and generate novel outputs (Zhang, et al., 2017). This fosters a richer and more varied set of generated sequences, which is beneficial for tasks requiring creative or exploratory outputs (Zhang, et al., 2017). Moreover, penalties can also enforce constraints on the generated sequences, such as controlling sequence length or

imposing structural constraints (Che, et al., 2017). For instance, penalties may penalize sequences that exceed a predefined length threshold or fail to adhere to specific syntactic or semantic rules (Che et al., 2017). By imposing such constraints, penalties ensure that the generated sequences meet certain criteria or standards, leading to outputs that are more aligned with the requirements of the task (Che et al., 2017).

In addition to promoting diversity and enforcing constraints, penalties in SeqGAN can also be optimized for specific performance metrics or objectives (Dai et al., 2018). These penalties are tailored to the particular requirements of the task and aim to direct the generator towards generating sequences that optimize the desired metric (Dai, et al., 2018). For example, penalties may prioritize the generation of sequences that exhibit high fluency, coherence, or relevance to a given context (Dai, et al., 2018). By incorporating such penalties into the loss function, SeqGAN can be trained to produce outputs that excel in specific aspects relevant to the task, leading to enhanced overall performance (Dai, et al., 2018). Furthermore, penalties can be used to penalize undesired behaviours or characteristics exhibited by the generator, such as mode collapse or poor sample quality (Dai, et al., 2018). By discouraging these behaviours, penalties encourage the generator to explore the output space more thoroughly and produce outputs of higher quality (Dai, et al., 2018).

It is essential to note that the design and implementation of penalties in SeqGAN involve careful consideration and experimentation (Zhang, et al., 2017). Researchers often explore various penalty formulations, weights, and combinations to achieve the desired balance between different objectives (Zhang, et al., 2017). Fine-tuning penalty parameters is an iterative process that requires experimentation and evaluation of the generator's performance across different penalty configurations (Zhang, et al., 2017). Moreover, the choice of penalties depends on the specific requirements of the task and the desired properties of the generated sequences. Consequently, penalties in SeqGAN are highly customizable and adaptable. It allows researchers to tailor them to the unique characteristics and objectives of their applications (Zhang et al., 2017)

2.6.1 Gradient Penalty

Gradient penalty (GP) is a regularization technique designed to stabilize the training of GANs by focusing on the discriminator. In GANs, the discriminator learns to distinguish between real and fake samples produced by the generator. If the discriminator becomes too confident too quickly, the generator might struggle to learn due to vanishing or overly harsh gradients. As a result, GP seeks to impose the Lipschitz continuity constraint on the discriminator or critic function, which can improve the GAN model's training stability and convergence.

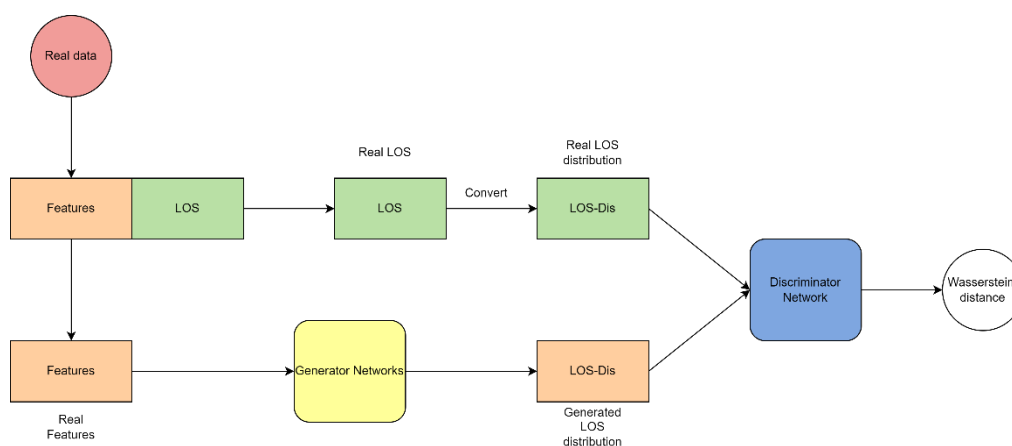


Figure 2.8: Wasserstein GAN with a gradient penalty for Length of Stay

The idea behind gradient penalties stems from the Wasserstein GAN (WGAN) framework, which reformulates the GAN objective as an optimization problem involving the Wasserstein distance between the real and generated distributions. In WGAN, the discriminator (or critic) function is required to be 1-Lipschitz continuous, meaning that the norm of its gradient is bounded by 1 everywhere. This constraint helps to ensure that the discriminator's output changes smoothly for its input, leading to better convergence properties.

To enforce the Lipschitz continuity condition, a gradient penalty term is added to the discriminator's loss function. This penalty term calculates the norm of the discriminator's gradient for its input and penalizes deviations from the desired Lipschitz constant (typically set to 1). The gradient penalty term can be formulated as

$$\lambda \cdot (\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2 \quad (2.5)$$

Where λ is a hyperparameter that controls the strength of the penalty term, $\nabla_{\hat{x}} D(\hat{x})$ the gradient of the discriminator's output with respect to its input $\|\cdot\|_2$ is the L2 norm. \hat{x} is a point sampled along a straight line between a pair of real and generated data points. This sampling ensures the penalty is applied across the data distribution (Kim, Park, & Hwang, 2018 ; Milne & Nachman, 2021).

Gradient penalties have been shown to improve the training stability and convergence of GANs, including SeqGAN with policy gradient. They help to mitigate issues such as mode collapse, gradient vanishing/exploding, and oscillatory behaviour during training. However, gradient penalties can also introduce additional computational overhead and may require careful hyperparameter tuning to achieve optimal performance (Zhang, & Gao, 2018 ; Jolicoeur-Martineau & Mitliagkas, 2020).

2.6.2 L1/L2 Regularization

L1/L2 Regularization prevents overfitting by including a penalty term in the loss function. Overfitting occurs when a model becomes closely tuned to the training data and loses the ability to generalize well to unseen examples. Their regularization techniques function by adding a penalty term to the loss function during training, which penalizes large weights in the model.

L1 regularization (Lasso) directly pushes some model weights toward zero, potentially leading to sparser models. L1 regularization involves adding a penalty term to the loss function that is proportionate to the weights' absolute value. In mathematical terms, it adds to the loss function the sum of the absolute values of the model weights multiplied by a regularization parameter. This encourages sparsity in the weight matrix, effectively pushing some weights to zero. L1 regularization is useful for feature selection since it yields sparse models.

On the other hand, L2 regularization (Ridge) generally shrinks the magnitude of weights without forcing them entirely to zero. L2 regularization includes a penalty term in the loss function that is proportional to the square of the weights. It mathematically adds the sum of the squares of the model weights to the loss function, multiplied by a regularization value. Unlike L1 regularization, L2 regularization penalizes large weights more smoothly, encouraging smaller but non-zero weights. L2 regularization is effective in preventing overfitting by spreading the weight values more evenly.

Table 2.8: Comparison of L1 and L2 regularization

L1 regularization	L2 regularization
<ul style="list-style-type: none"> • Sum of the absolute value of weights • Sparse solution • Multiple solutions • Built-in feature selection 	<ul style="list-style-type: none"> • Sum of square of weights • Non-sparse solution • One solution • No feature selection

The primary goal of L1 and L2 is to overfit prevention in the context of GAN training. They might indirectly contribute to greater stability by preventing the discriminator from becoming overly confident on a limited dataset. Their ease of implementation also makes L1/L2 regularization valuable as a baseline for assessing the relative impact of more complex penalty techniques.

2.6.3 Entropy Penalty

Entropy penalties, also known as entropy regularization, is a technique used in generative models, particularly in the context of Sequence Generative Adversarial Networks (SeqGAN) with policy gradient. The primary purpose of entropy penalties is to encourage the generator model to produce diverse and non-repetitive sequences during the training process.

The entropy penalty term is typically added to the generator's loss function, and it is designed to maximize the entropy of the generated output

distribution. Entropy is a measure of uncertainty or randomness in a probability distribution. By maximizing entropy, the generator is encouraged to explore a broader range of possible outputs, rather than collapsing to a limited set of outputs or modes. The entropy penalty term can be expressed as the following.

$$-\lambda \cdot H(G(z)) \quad (2.6)$$

where λ is a hyperparameter that controls the strength of the penalty term, $H(G(z))$ is the entropy of the generator's output distribution $G(z)$, given the input noise vector z . By minimizing the negative entropy, the generator is incentivized to produce output distributions with higher entropy. It will lead to more diverse and less repetitive sequences.

Entropy penalties are very beneficial for tackling the mode collapse problem, which is common in generative models such as GANs. Mode collapse happens when the generator learns to produce samples from a subset of the data distribution, resulting in a failure to capture the target distribution's complete diversity. By encouraging higher entropy in the generated outputs, entropy penalties can help mitigate mode collapse and promote better coverage of the data distribution.

However, it's important to note that while entropy penalties can improve diversity, they may also introduce irrelevant or incoherent outputs. Therefore, a balance needs to be struck between diversity and quality, often achieved by combining entropy penalties with other techniques like adversarial training, instance noise, or professor forcing.

2.6.4 Semantic Consistency Penalty

Semantic consistency penalties are regularization techniques specifically designed for generative models used in text generation tasks, like those involving Sequence Generative Adversarial Networks (SeqGAN) with policy gradient (Yu et al., 2017). Their core purpose is to promote semantic coherence

and consistency throughout the generated text (Zhang et al., 2021; Cifka et al., 2020).

Text generation models, even those as powerful as SeqGAN, may sometimes produce sequences with good local coherence but lack a broader semantic thread (Wiseman et al., 2018). This can result in abrupt topic deviations, nonsensical transitions, or even contradictions. Semantic consistency penalties address this by introducing a regularization term that penalizes deviations from the target semantic context (Xu et al., 2018).

Techniques to implement such penalties is using a pre-trained language model to estimate the likelihood of a generated sequence within a learned semantic context (Holtzman et al., 2020). Sequences with low likelihood are penalized, encouraging consistency. Additionally, similarity measures like cosine similarity or word embeddings can assess the semantic closeness between the generated text and the desired context (a prompt, topic, etc.) (Wieting et al., 2019). Deviations from expected similarity incur penalties. Finally, in models utilizing attention mechanisms, the penalty may promote the alignment of attention weights with the intended semantic focus (Bahdanau et al., 2015).

By adding a semantic consistency penalty to the loss function, the generator is nudged towards outputs that are not just locally coherent, but maintain global consistency with the target (Li et al., 2017). This is particularly valuable in applications like storytelling, dialogue systems, or creative writing, where semantic consistency holds high importance. Importantly, achieving a balance between semantic consistency and other desired qualities like diversity and novelty is crucial (Cifka et al., 2020). Too strong penalties might result in overly repetitive or safe outputs. Thus, semantic consistency penalties often work in conjunction with other regularization methods or objectives to strike the right balance (Zhang et al., 2021).

Table 2.9: Comparison of various regularization and penalty techniques

Penalties	Primary Focus	Mechanism	Impact on Stability	Impact on Quality	Implementation Complexity
Gradient Penalty	<ul style="list-style-type: none"> • Discriminator Stability 	<ul style="list-style-type: none"> • Penalizes large gradients 	<ul style="list-style-type: none"> • Strong stabilization effect 	<ul style="list-style-type: none"> • Indirectly improves quality due to stability 	<ul style="list-style-type: none"> • Moderate (requires adaptation to text)
L1/L2 Regularization	<ul style="list-style-type: none"> • Overfitting Prevention 	<ul style="list-style-type: none"> • Penalizes model weights 	<ul style="list-style-type: none"> • Indirectly improve stability 	<ul style="list-style-type: none"> • Less direct impact on quality 	<ul style="list-style-type: none"> • Simple
Entropy Penalties	<ul style="list-style-type: none"> • Output Variety 	<ul style="list-style-type: none"> • Modifies reward signal or loss to encourage diversity 	<ul style="list-style-type: none"> • Less direct impact on stability 	<ul style="list-style-type: none"> • Directly promotes diversity, might need balancing 	<ul style="list-style-type: none"> • Moderate to high (metric choice is crucial)
Semantic Consistency Penalty	<ul style="list-style-type: none"> • Semantic Realism 	<ul style="list-style-type: none"> • Penalizes semantically dissimilar outputs 	<ul style="list-style-type: none"> • Less direct impact on stability 	<ul style="list-style-type: none"> • Strong potential for realism if the metric is well-defined 	<ul style="list-style-type: none"> • High (requires NLP techniques)

Table 2.9 shows the key attributes of various penalties that are relevant for text-generating while using SeqGAN. Gradient Penalty (GP) explicitly promotes discriminator stability, a core concern in GAN training. Successful usage in SeqGAN requires careful adaptation for sequential language data. Classic L1/L2 regularization provides an overfitting prevention baseline. For scenarios where mode collapse is prominent, entropy/diversity penalties directly encourage broader output exploration. Semantic consistency penalties aim to enhance realism but necessitate defining robust NLP-based similarity metrics. Finally, data augmentation penalties promote strength to input variation, with their effectiveness depending on defining task-suitable transformations.

Table 2.10: Implication of Regulatory Mechanisms on Model Performance

Regulatory Mechanism	Advantages	Disadvantages	Impact on Model Complexity
Gradient Penalties	<ul style="list-style-type: none"> Improves model stability and convergence. Helps mitigate issues like mode collapse. 	<ul style="list-style-type: none"> Introduce computational complexity. Require careful tuning of the hyperparameter 	<ul style="list-style-type: none"> Increases model complexity due to the additional computations for gradient norms.
L1/L2 Regularization	<ul style="list-style-type: none"> Produce simpler, more interpretable models. L1 yields sparse models, which are beneficial in high-dimensional settings. 	<ul style="list-style-type: none"> L2 does not produce sparse models Will be a drawback in scenarios where feature selection is crucial. 	<ul style="list-style-type: none"> Generally increases training stability Lead to underfitting if over-penalized.
Entropy Penalty	<ul style="list-style-type: none"> Promotes diversity in the outputs Lead to more robust learning, especially in reinforcement learning contexts. 	<ul style="list-style-type: none"> Too much entropy will lead to overly random outputs May decrease the utility of the model's outputs. 	<ul style="list-style-type: none"> Slightly increases model complexity due to the need to calculate entropy and adjust training procedures.
Semantic Consistency Penalty	<ul style="list-style-type: none"> Ensures the outputs are semantically consistent with the inputs 	<ul style="list-style-type: none"> Apply semantic consistency checks will be difficult Require complex architectures. 	<ul style="list-style-type: none"> Significantly increases model complexity. Requires auxiliary models or systems to assess consistency.

Table 2.10 highlights the implication of various regulatory mechanisms on Model Performance in terms of their definitions, primary functions, benefits, drawbacks, and their impact on model complexity and performance.

Fundamentally, the ideal penalty choice depends on the specifics of the SeqGAN implementation, dataset, and goals, as outlined by the evaluation metrics. Penalties offering the greatest quality potential often come with higher complexity. Investigating research adapting these techniques to text-based GANs similar to SeqGAN will be essential for informed decision-making.

2.7 Summary

The literature review has illuminated the multifaceted field of Generative Adversarial Networks (GANs), with a focus on their evolution into Sequence Generative Adversarial Networks (SeqGANs) and the subsequent integration with policy gradient methods. This synthesis reveals that while GANs have set the stage for profound advancements in generative models, SeqGANs have taken the baton further by navigating the complicated pathways of text generation.

From the comparative overview of GANs and SeqGANs to the analysis of policy gradient methods, this review has highlighted the nuances and complexities inherent in the field. Architectural innovations have been shown to enhance the generative capabilities of these networks, addressing the challenges of creating coherent and contextually rich textual content. Evaluation metrics and the role of regulatory mechanisms and penalties have also been important in refining the training process and output quality of SeqGANs.

This literature review has not only shed light on the current state of generative models but also underscored the significant gaps and opportunities for further research. The knowledge gaps identified point towards the necessity of a robust framework capable of harnessing the full potential of SeqGANs, especially in the context of text generation.

As AI continues to evolve, the implications of these developments extend beyond theoretical research, promising to redefine the landscape of automated content generation. The findings of this review lay the groundwork for exploring novel methodologies and serve as a beacon for future investigations aiming to push the boundaries of generative AI.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter emphasizes the systematic approach taken to ensure the project's objectives are achieved.

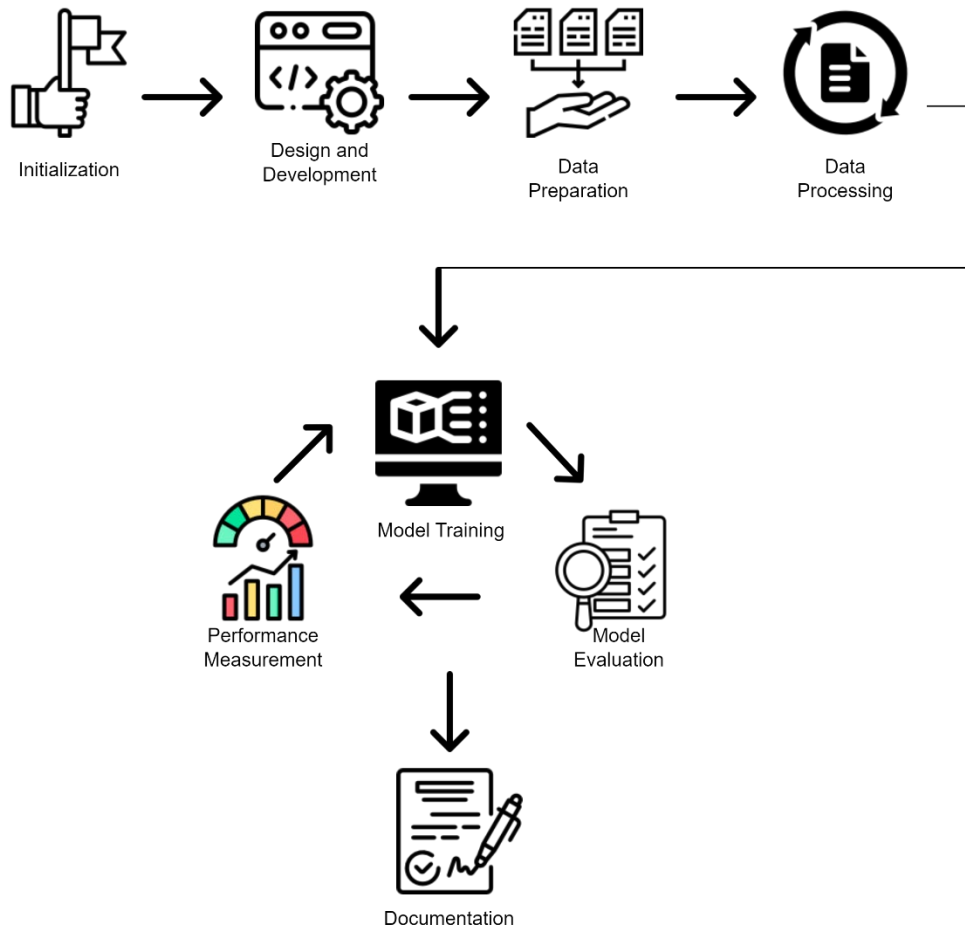


Figure 3.1: Summary of Project Workflow

Figure 3.1 summarises the project workflow of the project. The project begins with the Initialization Stage. During this phase, the computational environment is configured with all the required software, and the hardware specifications are set to accommodate the demands of the SeqGAN with policy gradient model.

Following the initial setup, the project transitions into the design and development phase. Here, the SeqGAN enhanced model's architecture is crafted,

integrating advanced neural networks like GRU and incorporating attention mechanisms for enhanced text generation capabilities. Alongside the architectural design, regularization method, gradient penalty is selected and fine-tuned with a particular emphasis on optimizing the balance between exploration and exploitation.

Next is the Data Collection and Preprocessing Stage, where data central for training the SeqGAN model is gathered, curated, and preprocessed. This involves curating a dataset from selected sources, which in this case might be a repository of Chinese poetry. It ensures that the data aligns with the project's aims. The collected data is then preprocessed, involving cleaning, tokenization, and possibly vectorization, to prepare it for training the SeqGAN model.

During the Model Training stage, 2 models are being implemented which are a baseline model and an enhanced model. The baseline model referring to the original model that was found in the GitHub repository without doing any enhancement for the model architecture design and hyperparameter tuning. During this phase, two settings are being used to test the sensitivity of the hyperparameter values.

During the Evaluation Stage, the performance of the baseline model and enhanced model is carefully measured using a set of predefined metrics. During this phase, both models are trained and evaluated using a variety of metrics designed to gauge both the syntactic and semantic quality of the generated text. This includes BLEU scores for evaluating the quality of the generated text, adversarial loss for measuring the performance of the model and NLL measures for how well the model predicts and generates a sequence that is close to the real samples.

Upon generating satisfactory results, the project shifts focus to performance optimization and fine-tuning. It is a critical phase where the SeqGAN model's output is meticulously analyzed, and adjustments are made to improve its efficacy in generating high-quality text.

Finally, the workflow concludes with the Monitoring and Documentation Stage. Here, both of the trained model being compared and analyse. A comprehensive documentation is maintained to capture the development process, results, and insights gained throughout the project.

3.2 Software and Tools

3.2.1 Tools

3.2.1.1 PyCharm

PyCharm is an Integrated Development Environment (IDE) for Python programming. It will serve as the primary development platform due to its comprehensive coding assistance, intelligent code editor, and debugging features. It was used for writing, testing, and debugging the Python code that constitutes the SeqGAN and policy gradient implementations. Its intelligent editor provides code completion, syntax highlighting, and on-the-fly error detection, which are invaluable for rapid development cycles.

3.2.1.2 Draw.io

Draw.io is used for creating flowcharts, process diagrams, and architectures which will be important in planning the SeqGAN model's structure. Diagrams created using Draw.io will be used in project documentation to illustrate the data flow and operational logic of the policy gradient methods. It provides visual representations that help in making the project's technical aspects accessible and easier to understand for reviewers and collaborators.

3.2.1.3 GitHub

GitHub is a platform for version control and collaboration. GitHub provides tools for branching, merging, and pulling requests through hosting the project repository. GitHub's issues and project boards will help keep track of tasks, enhancements, and bugs, which is essential for managing complex development projects with potentially multiple collaborators.

3.2.2 Hardware Environment

The computing system used is a Lenovo model that runs a 64-bit version of Microsoft Windows 11 Home Single Language. This operating system was

selected for its stability and broad support for the necessary development tools and libraries. Besides that, the system's performance is a 13th Gen Intel(R) Core(TM) i7-13700HX CPU with a base speed of 2.00 GHz. The processor boasts 16 cores and can handle 24 logical processes simultaneously. This will offer significant multitasking capabilities for parallel computations during model training.

Next, the system is equipped with a substantial 32 GB of installed physical RAM. The RAM will ensure the smooth execution of multiple large-scale operations and datasets concurrently without constricting the computational processes. Storage is handled by a device that offers ample space and speed for data-intensive tasks to access and process large datasets quickly and efficiently. Table 3.1 will summarize the overall hardware components used in this study.

Table 3.1: Hardware specifications

Component	Specification
OS Name	Microsoft Windows 11 Home Single Language
Processor	Intel(R) Core(TM) i7-13700HX CPU, 2.00 GHz, 16 Cores, 24 Logical Processors
BIOS Version	LENOVO CNKCN38W, 20/6/2023
System Type	x64-based PC
RAM	32.0 GB Installed Physical Memory
Total Physical Memory	31.7 GB
Total Virtual Memory	33.7 GB
Storage	953.86 GB SAMSUNG MZVL21T0HCLR-00BL2

3.2.3 Software Environment

Anaconda, a popular package and environment management system, is used for the development and execution of the SeqGAN-based text generation project to

construct a specialized software environment. The environment encapsulates a suite of libraries and frameworks tailored for deep learning tasks.

Table 3.2: Software specifications

Software	Version
Python	3.8
TensorFlow	2.11
Keras	2.3.1
NumPy	1.24.1
Pandas	1.5.3
Matplotlib	3.8.4
SciPy	1.5.3
CUDA Toolkit	12.4

Table 3.2 shows the software specifications that will be used in this project. The project's software environment is underpinned by Python 3.8. It was used as the primary programming language due to its rich ecosystem of libraries and widespread use in scientific computing. TensorFlow 2.11 is employed to facilitate the construction and training of the complex neural networks at the core of SeqGAN. Keras 2.3.1 was integrated with TensorFlow and offers a streamlined, high-level neural network API that will simplify deep learning programming tasks.

For mathematical computations and operations of machine learning, NumPy 1.24.1 can provide support with its array of objects and mathematical functions. Pandas 1.5.3 is utilized for its data manipulation expertise, which helps in preparing the textual datasets for the model. Visualization of data and model performance is handled by Matplotlib 3.8.4 to produce a wide range of static, animated, and interactive visualizations.

Scientific and technical computing tasks that involve optimization and linear algebra will be managed by SciPy 1.5.3. Its modules are designed to work efficiently with NumPy arrays and provide user-friendly interfaces to numerical

routines. In addition, CUDA Toolkit 12.4 allows TensorFlow to perform high-speed computations, thus it accelerates the training and evaluation of the SeqGAN models.

3.3 Work Plan

The project's timeline, key milestones, and deliverables are outlined in the Work Breakdown Structure (WBS) and Gantt chart. The Gantt chart presents a clear work plan with a project start date, along with completion dates for each phase of the project.

Work Breakdown Structure (WBS)

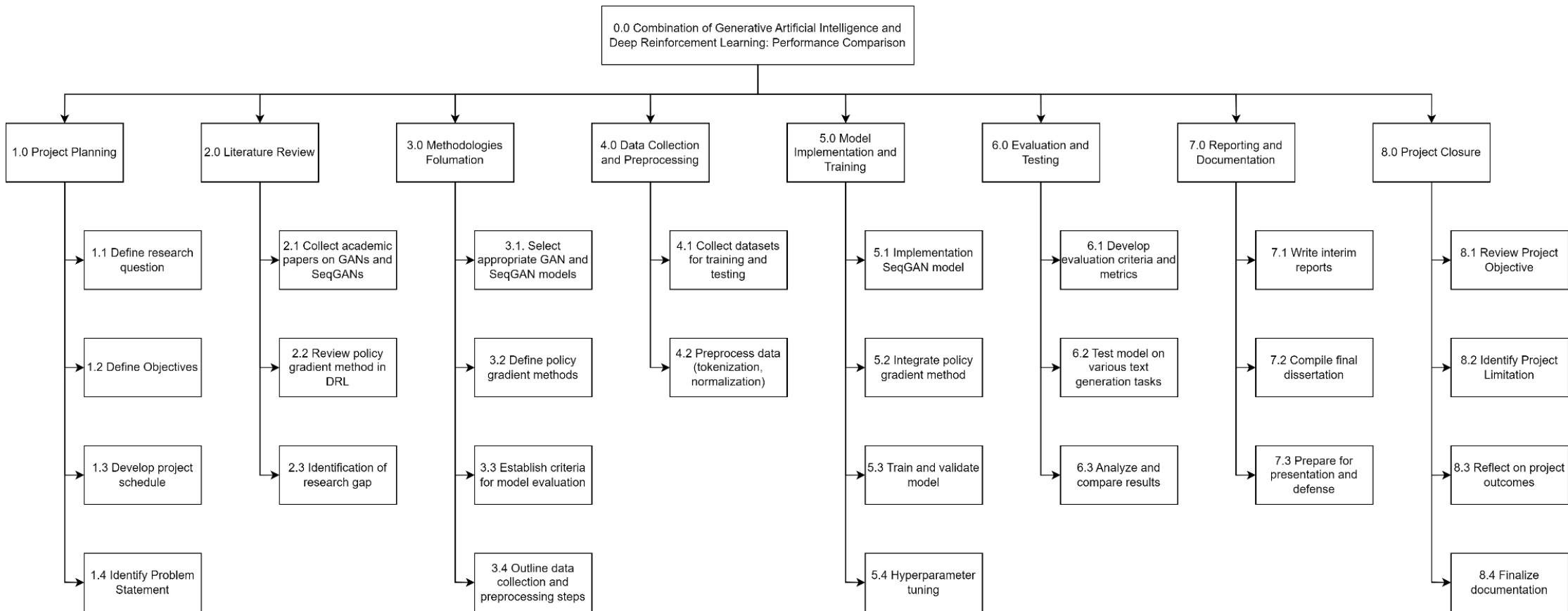


Figure 3.2: Project Work Breakdown Structure (WBS)

Gantt Chart

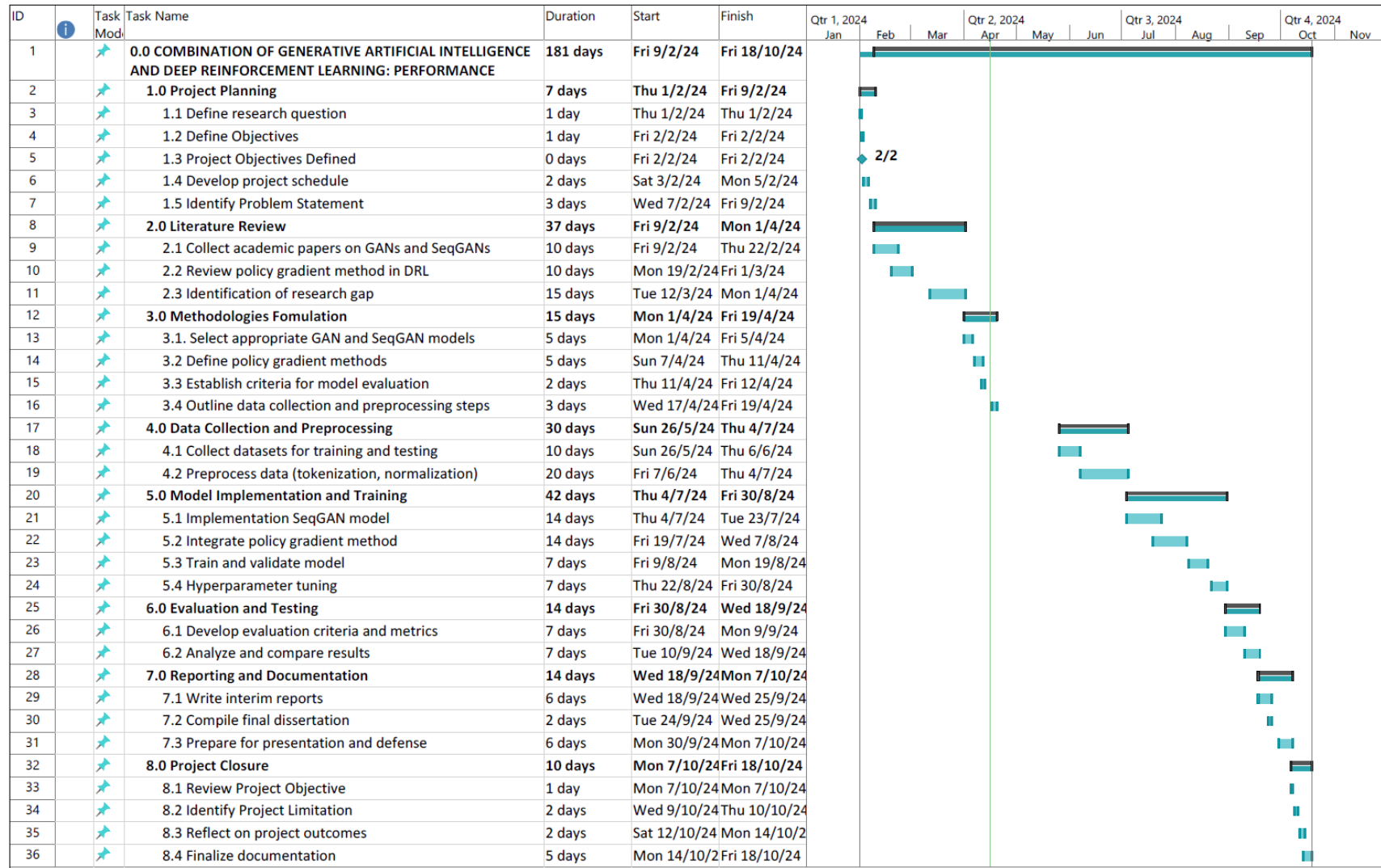


Figure 3.3: Gantt Chart

CHAPTER 4

PROJECT INITIAL SPECIFICATION

4.1 Introduction

This chapter provides a detailed overview of the initial specifications for the SeqGAN model, focusing on architectural enhancements, integration of regularization mechanisms, data handling processes, and evaluation strategies. This chapter sets the stage for transforming theoretical concepts into a practical framework, ensuring the model's effectiveness in generating high-quality, context-rich textual content. It lays the groundwork for the project's development and subsequent performance assessment for achieving the goal of blending AI sophistication with the nuance of Chinese poetry.

4.2 Data Collection

This project will utilize a dataset from the Chinese-poetry GitHub repository (<https://github.com/chinese-poetry/chinese-poetry>), which contains a comprehensive collection of classical Chinese poetry. This dataset is an extensive anthology of poems that encapsulate a wide array of emotions, themes, and styles. This can provide a rich linguistic environment for training the generative model.

volume	title	author	body
1_1	还陕述怀	李世民	慨然抚长剑，济世岂邀名。星旂纷电举，日羽肃天行。遍野屯万骑，临原驻五营。登山麾武节，背水纵神兵。在昔戎戈动，今来宇宙平。
1_2	临洛水	李世民	春搜驰骏骨，总辔俯长河。霞处流萦锦，风前漾卷罗。水花翻照树，堤兰倒插波。岂必汾阴曲，秋云发棹歌。
1_3	望终南山	李世民	重峦俯渭水，碧嶂插遥天。出红扶岭日，入翠贮岩烟。叠松朝若夜，复岫晓疑全。对此恬千虑，无劳访九仙。
1_4	于北平作	李世民	翠野驻戎轩，卢龙转征旆。遥山丽如绮，长流萦似带。海气百重楼，岩松千丈盖。兹焉可游赏，何必襄城外。
1_5	辽城望月	李世民	玄兔月初明，澄辉照辽碣。映云光暂隐，隔树花如缀。魄满桂枝圆，轮亏镜彩缺。临城却影散，带晕重围结。驻眸俯九都，停观妖氛灭。

Figure 4.1: Sample of Chinese Poetry

Figure 4.1 represents a small snippet from the dataset. It illustrates the type of content that the model will be trained on. The dataset consists of 1.5 million Chinese poems. This poetry was separated by different timelines and authors.

4.3 Data Processing

The initial step involves sanitizing the text data, which includes stripping away any irrelevant characters or formatting. This step is crucial in maintaining the

linguistic integrity of the original poetry while ensuring that the data is conducive to the learning algorithm's requirements. Given the intricacies of the Chinese language, particularly in classical poetry, tokenization will be performed with an understanding of the linguistic and poetic context. The goal is to convert the corpus into an appropriate format for the model to process, balancing the preservation of literary elements with the technical demands of tokenization.

秦川雄帝宅 函谷壮皇居 绮殿千寻起 离宫百雉馀
 连甍遥接汉 飞观迥凌虚 云日隐层阙 绮风烟出疏
 岩廊罢机务 崇文聊驻辇 玉匣启龙图 金绳披凤篆
 韦编断仍续 缥帙舒还卷 对此乃淹留 观歎案坟典
 出移步词林 停舆欣武宴 雕弓写明月 骏马疑流电
 虚惊雁落弦 啼猿悲急箭 阅赏诚多美 乃于兹忘倦
 鸣笳临乐馆 眺听欢芳节 弦急管韵朱 清歌凝白雪
 凤彩肃来仪 玄鹤纷成列 兹去郑卫声 雅音方可悦
 芳辰追逸趣 多禁苑信奇 汉桥形通上 接云峰势危
 隐烟霞交映 花鸟自参差 何如肆辙迹 赏万里瑶池
 飞芳去盖园 兰橈游翠渚 日彩萍间乱 风荷处香举
 川桂楫满中 弦歌振长屿 岂必汾河曲 宴欢方为所
 日阙落双昏 舆回九重暮 烟长散初碧 月皎澄轻素

Figure 4.2: Sample of Chinese Poetry after tokenization

Subsequently, the tokenized text will be transformed into numerical vectors. Decisions regarding the use of word embeddings versus one-hot encoding will be made based on the model's architecture and the nature of the text. This vectorization process is instrumental in facilitating the model's comprehension of the data. Post-vectorization, the corpus will be segmented into sequences with uniform length. This standardization is critical for the consistent training of the SeqGAN model, allowing for each input sequence to be fed into the model systematically. To enhance the SeqGAN's learning efficiency, the data may undergo normalization procedures, such as lowercasing, to standardize the input and decrease the complexity of the model's vocabulary.

4.4 SeqGAN Model Enhancement

4.4.1 GRU Layer Integration

Recognizing the important role of memory in sequence generation tasks, the improved SeqGAN architecture will incorporate GRU (Gated Recurrent Units) layers. These advanced recurrent layers are renowned for their superior capacity to capture long-term dependencies within sequential data. The depth of these networks is critical; deeper layers are synonymous with a model's capability to understand and encode more complex patterns and dependencies, an attribute essential for generating coherent and extensive textual content.

By implementing GRU, we anticipate a substantial enhancement in the model's ability to produce text that is not only syntactically and grammatically correct but also contextually rich. This holds especially true for sequences that demand continuity over extended narrative arcs, thus addressing one of the significant challenges faced by the current SeqGAN models.

4.4.2 Transformer Intergration

The Transformer architecture is a highly parallelizable approach that makes sequence-generating activities more efficient by utilizing self-attention processes. Transformers handle sequences more effectively than recurrent designs like GRU and LSTM, which process tokens sequentially. They do this by responding to all tokens at the same time, capturing long-range relationships in a sequence without the requirement for recurrent connections.

The Transformer consist do two key components: self-attention mechanism and positional encoding. The self-attention mechanism enables the model to focus on specific parts of the input sequence when predicting the next token. It assigns different attention scores to tokens, allowing the model to weigh the importance of each token about others in the sequence. This capability is crucial for handling long-range dependencies and context maintenance. On the other hand, positional encoding is used to maintain information about the token positions within the sequence since the Transformer does not have an inherent sense of order. This helps the model understand the structure of the sequence, which is essential for generating coherent text.

By integrating Transformers into the SeqGAN model, the project aims to improve the generation of high-quality sequences, particularly in generating classical Chinese poetry, by leveraging the model's ability to handle long-range dependencies and parallelization.

4.4.3 Reward Structure

In the context of SeqGAN, the discriminator's feedback is important in guiding the generator's learning process. To leverage this, a sophisticated reward structure is proposed that incorporates intermediate rewards. These rewards are dispensed at various checkpoints based on the discriminator's evaluation of the generated sequences. This structure aims to provide more granular feedback to the generator that enables it to make more informed updates to its policy network.

Furthermore, the introduction of intermediate rewards presents an opportunity to refine the generator's learning curve further. Unlike traditional reinforcement learning setups where rewards are sparse and only received at the end of an episode, intermediate rewards provide frequent and actionable feedback. This can accelerate the generator's learning process, helping it to quickly identify and reinforce successful strategies for sequence generation. It also helps mitigate the sparse reward problem typically associated with training generators in GAN frameworks.

4.5 Optimization of SeqGAN

In the optimization of the SeqGAN model, the Wasserstein Loss with Gradient Penalty (WGAN-GP) will be implemented to improve the stability of the training process for GANs by using the Earth Mover's distance. When it combines with the gradient penalty, it enforces 1-Lipschitz continuity.

$$L = E_{\tilde{x} \sim P_g} [D(\tilde{x})] - E_{x \sim P_r} [D(x)] + \lambda E_{\hat{x} \sim P_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \quad (4.1)$$

where P_g and P_r are the distributions of generated data and real data respectively. \hat{x} is evenly sampled along straight lines between pairs of points collected from generated and real data distributions. λ represents the penalty coefficient.

Since SeqGAN generates sequences incrementally and relies on the discriminator's feedback at each step, Wasserstein loss will be used due to its stable and robust loss function. It can make the feedback more meaningful and consistent, which is critical for effective learning in a policy gradient setup.

4.6 Evaluation Metrics and Performance Analysis

Various metrics are chosen to evaluate the performance of SeqGAN when the model is used to generate sequences. The automated metrics will be used in this study will be the BLEU score and Oracle NLL.

The BLEU score will evaluate how closely the generated text matches a reference by comparing the overlap of n-grams. It is widely used in text generation tasks to gauge syntactic consistency. However, BLEU has limitations in capturing semantic coherence, as it may give high scores to texts that are not meaningful.

Besides that, Oracle NLL measures how well the generated sequence matches the real data distribution. Lower NLL values indicate that the generated sequence closely resembles the reference, and thus, better model performance.

Finally, the performance of the model (generator and discriminator) will be evaluated through adversarial loss. During adversarial training, the generator's performance is judged based on its ability to produce realistic sequences that can "fool" the discriminator. A lower generator loss indicates an improvement in generating realistic data. While the discriminator measures its ability to differentiate between real and generated data. A smaller loss indicates stronger distinction, but extremely low values may suggest that the discriminator is unduly dominating, thus impeding the generator's learning.

4.7 Summary

Table 4.1: Overview of the Proposed Solution

Feature	Baseline Model	Enhanced Model
Generator Model	<ul style="list-style-type: none"> Generates the next token based on the current state (LSTM). 	<ul style="list-style-type: none"> Use GRU architecture Implement transformer with attention mechanisms
Discriminator Model	<ul style="list-style-type: none"> Binary classifier LSTM that discriminates real vs. generated sequences. 	<ul style="list-style-type: none"> Same as original, but lower learning rate
Reward Structure	<ul style="list-style-type: none"> Sparse and binary, given at the end of sequence generation. 	<ul style="list-style-type: none"> Same as original, but include intermediate rewards based on discriminator feedback at checkpoints.
Training Stability	<ul style="list-style-type: none"> Can suffer from instability due to sparse rewards and high variance in policy updates. 	<ul style="list-style-type: none"> Improved stability using gradient penalty to enforce Lipschitz constraint.

Table 4.1 presents an overview of the proposed method compared with the original SeqGAN method that was implemented in the repository found in GitHub.

In summary, this chapter outlined the project's initial planning for enhancing the SeqGAN model to generate classical Chinese poetry. By integrating deep learning innovations such as GRU layers and Transformer with attention mechanisms, the project seeks to harness the vast potential of SeqGAN.

The initial planning is underpinned by a thoughtful data collection and preparation process, ensuring that the model is fed with high-quality and context-rich poetic content. This is coupled with a meticulous evaluation strategy that employs automated metrics, designed to provide a holistic assessment of the model's performance.

CHAPTER 5

RESULT AND DISCUSSION

5.1 System Performance

In this research, the results are compared between two SeqGAN models: baseline and enhanced models based on their key performance. The model's performance is enhanced with two major improvements which are GRU integration and a Transformer architecture in SeqGAN for poem generation. Both models have been tested using the same hyperparameter configurations to ensure a fair comparison.

To assess the model's output, both quantitative and qualitative measures are used. The analysis focuses on comparing the baseline SeqGAN and the enhanced version in terms of generator and discriminator loss, Oracle Negative Log-Likelihood (NLL), BLEU score, and performance across different hyperparameter configurations.

5.1.1 Quantitative Metric

5.1.1.1 Hyperparameter Configuration

The hyperparameters used in both models are summarized in Table 1. The baseline model uses standard adversarial training with LSTM, while the enhanced model incorporates GRU and Transformers. The model may suffer from issues like mode collapse, where it generates limited variations of data that the discriminator can't classify as fake. So a gradient penalty was applied to improve training stability by enforcing Lipschitz continuity which helps mitigate mode collapse and stabilize the model.

Table 5.1: Hyperparameter configurations

Hyperparameters	Values	
	Setting 1	Setting 2
Rounds	50	100

Generator Pretraining Steps (g_pretrain_steps)	100	120
Discriminator Pretraining Steps (d_pretrain_steps)	50	50
Generator Steps per Round (g_steps)	3	5
Discriminator Steps per Round (d_steps)	1	3
Generator Learning Rate	0.01	0.01
Discriminator Learning Rate	0.0001	0.0001
Update Rate	0.8	0.8
Vocabulary Size	6915	6915
Batch Size	32	64

5.1.1.2 Adversarial Training Performance

Generator Adversarial Training Loss

The performance of the generator during adversarial training is evaluated based on how well it improves the quality of the data that can be determined by the discriminator. Lower generator loss indicates that the generator is improving in creating realistic data. If the loss is consistently high, this indicates the generator struggles to produce convincing data and the discriminator can easily differentiate between real and generated data.

Table 5.2: Generator Adversarial Training Loss Result

Settings	Epoch	Generator Loss	
		Baseline	Enhanced
1	20	5.7422	2.5078
	60	4.9757	0.1451
	100	4.7949	0.0186
2	20	5.7490	3.235
	60	4.9777	0.959
	120	4.7621	0.131

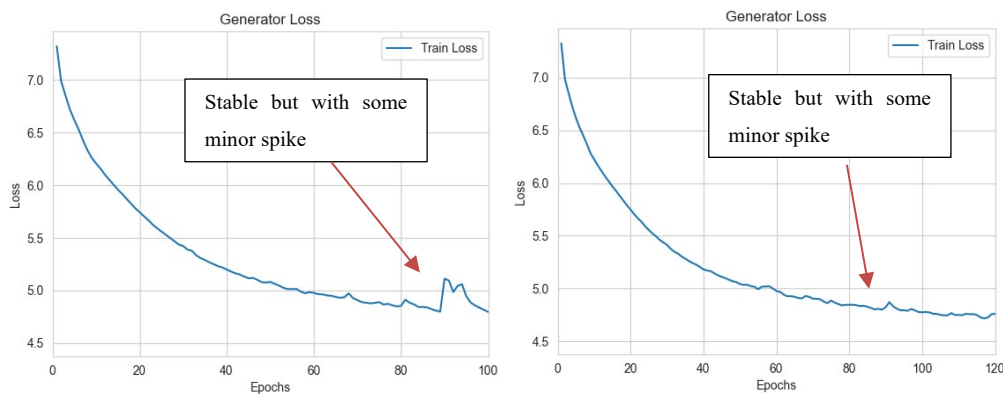


Figure 5.1: Generator loss of baseline model using setting 1 and setting 2

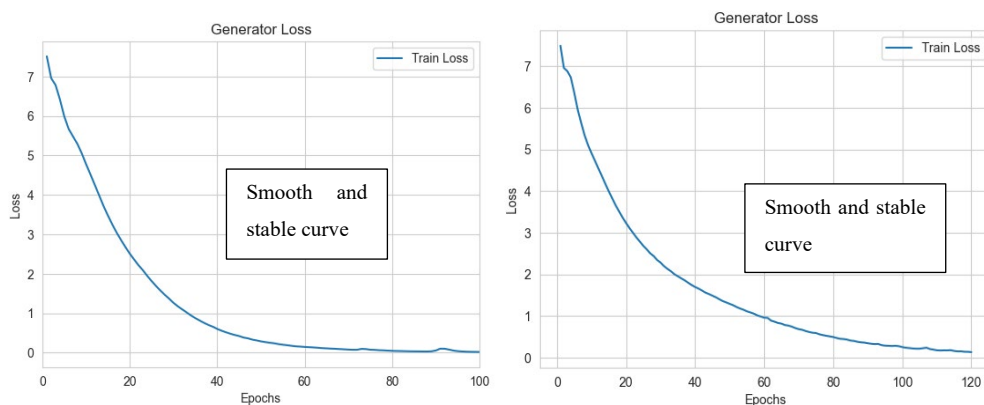


Figure 5.2: Generator loss of enhanced model using setting 1 and setting 2

In setting 1, the baseline model started at 5.7422, and ended at 4.7949, while the enhanced model started at 2.5078 and improved to 0.0186. In setting 2, the baseline begins at 5.7490 and finishes at 4.762, while the enhanced model begins at 3.235 and significantly improves to 0.131 at the end. The enhanced model showed a much steeper decrease in generator loss, which indicates faster and more effective learning for the generator. GRU allows the generator to capture long-term dependencies more efficiently in sequential data and retain relevant information from earlier in the sequence which makes the learning process more efficient. GRU's gating mechanisms (reset and update gates) reduce the vanishing gradient problem, which allows the model to propagate important information over long sequences. This enables the generator to make better updates during training, resulting in faster convergence of the generator

loss. These results explore the cooperation effects between various SeqGAN architectures. The GRU's ability to efficiently handle long-range dependencies accelerates the generator's learning process, thereby improving text generation quality as the model learns more from each training iteration.

Discriminator Adversarial Training Loss

The discriminator's performance measures how well it can distinguish between real and generated data during the adversarial training process. Lower discriminator loss means it is effectively distinguishing real from fake data. However, too low a loss may indicate that the discriminator is dominating the training and making it hard for the generator to improve.

Table 5.3: Discriminator Adversarial Training Loss Result

Settings	Epoch	Discriminator Loss	
		Baseline	Enhanced
1	30	0.6770	0.5267
	50	0.6612	0.0318
2	30	0.5629	0.5628
	50	0.5435	0.5314

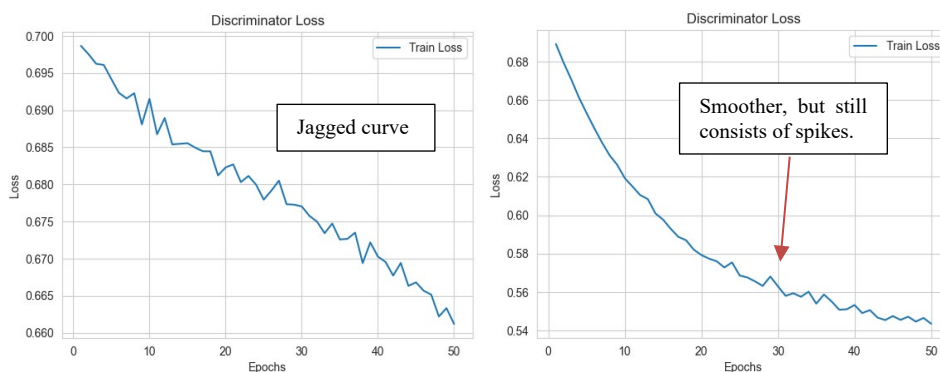


Figure 5.3: Discriminator loss of baseline model using setting 1 and setting 2

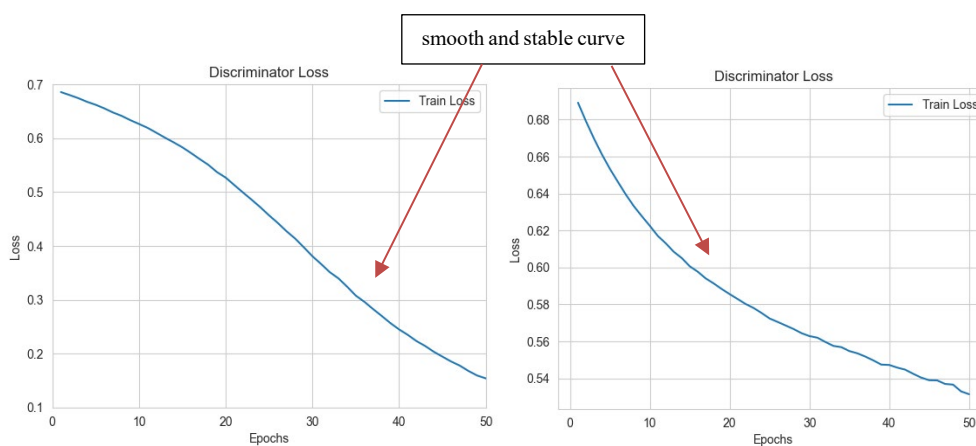


Figure 5.4 Discriminator loss of enhanced model using setting 1 and setting 2

In setting 1, the baseline model decreases slightly from 0.6770 to 0.6612, while the enhanced model decreases dramatically from 0.5267 to 0.0318. In setting 2, the baseline drops slightly from 0.5629 to 0.5435, while the enhanced model decreases from 0.5628 to 0.5314. The performance of the discriminator in the baseline model is not stable compared to the enhanced model. The enhanced model shows a smooth curve which is more stable while the baseline model consists of a lot of spikes. This stability is a result of the gradient penalty, which regularizes the discriminator, preventing it from learning too quickly and overpowering the generator. Without this penalty, the baseline model's discriminator shows spikes in its loss, a sign of instability in adversarial training. The results emphasize regulatory mechanisms like gradient penalties. These mechanisms help stabilize the adversarial training process, ensuring that the discriminator and generator learn at balanced rates, resulting in more consistent model improvements.

The gradient penalty regularizes the discriminator by ensuring the output changes gradually concerning small changes in the input. It penalizes the discriminator if the gradients of its predictions become too large or too small. Without a gradient penalty, the discriminator in the baseline model may learn too quickly and become overly confident in distinguishing real from fake data. This will lead to spikes in its loss function if the complexity of the generator is high. These sharp changes can destabilize the entire adversarial training process. In contrast, the gradient penalty in the enhanced model keeps the discriminator's

updates more controlled and gradual, leading to smoother and more stable learning curves.

5.1.1.3 Oracle NLL

Generator Oracle NLL

The generator Oracle NLL measures how well the generator is performing in terms of generating sequences that resemble the real data distribution. Lower NLL values indicate better performance and this also means the generator is producing sequences that are closer to the true data.

Table 5.4: Generator Oracle NLL result

Settings	Epoch	Generator Oracle NLL	
		Baseline	Enhanced
1	20	11.5810	6.1021
	60	11.7417	4.1472
	100	11.7869	2.8082
2	20	11.7015	5.4534
	60	11.6649	3.3654
	120	11.6540	2.347

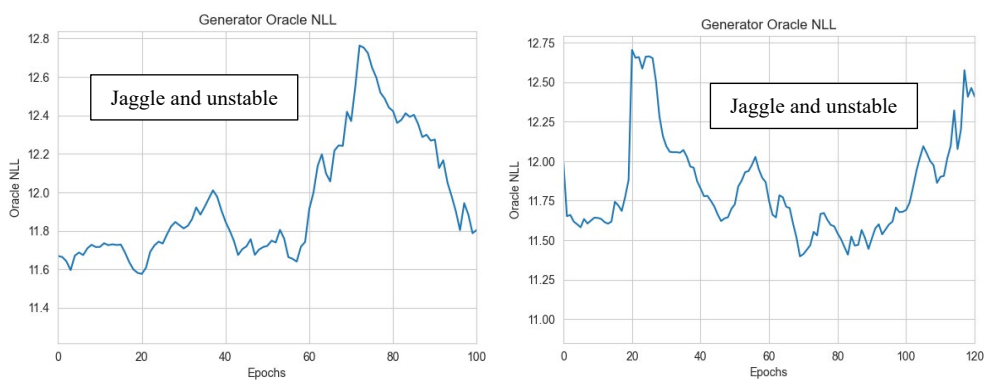


Figure 5.5: Generator Oracle NLL of baseline model using setting 1 and setting 2

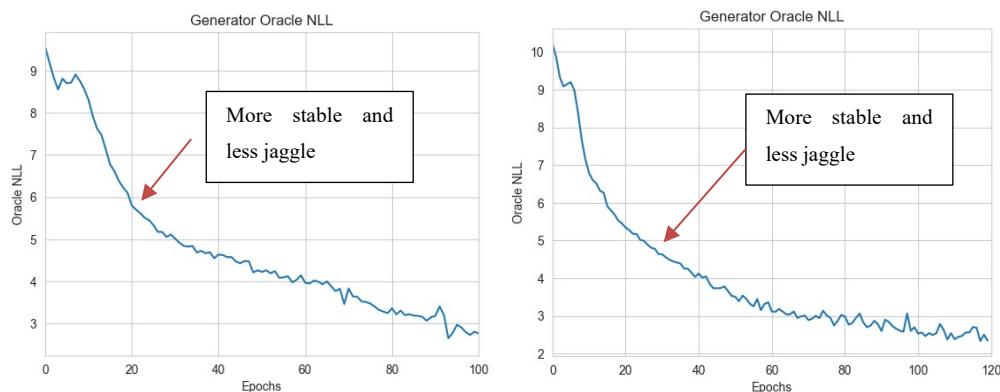


Figure 5.6: Generator Oracle NLL of enhanced model using setting 1 and setting 2

In Setting 1, the baseline model increased slightly from 11.5810 to 11.7869, while the enhanced model decreased significantly from 6.1021 to 2.8082. In Setting 2, the baseline model decreased from 11.7015 to 6540 while the enhanced model decreased from 5.4534 to 2.347. the baseline model shows a significant fluctuation in both settings 1 and 2, which indicates the instability of the generator. The Lower Oracle NLL scores indicate that the enhanced model is generating sequences that are closer to the true data distribution, which indicates better quality output.

The enhanced model's significantly lower Oracle NLL shows its improved ability to generate sequences that are closer to the real data distribution. This improvement can be attributed to the integration of GRU and Transformer architectures. The GRU's gating mechanism helps the generator retain important information from earlier parts of the sequence, while the Transformer's attention mechanism allows the model to focus on crucial sequence elements such as rhyme, line length, and semantic coherence. GRU allows the model to better retain long-term dependencies by efficiently controlling what information to forget and what to retain in sequential data. This ability to manage long-term dependencies prevents the vanishing gradient problem and ensures the important information from earlier steps in the sequence is preserved during the learning process. Besides that, the Transformer's attention mechanism enables the model to focus on the most important elements of the sequence (rhyme, line length, and parallelism) and direct its resources toward understanding the key contextual relationships

between data points. Together, these architectural improvements enable the generator to produce higher-quality sequences that better match the true data distribution, thus reducing the NLL significantly.

Discriminator Oracle NLL

The discriminator Oracle NLL is used to evaluate the discriminator's ability to distinguish between real and generated sequences. It measures how well the discriminator can predict whether a given sequence is from the real data distribution (Oracle) or generated by the generator. A lower Oracle NLL indicates better performance by the discriminator.

Table 5.5: Discriminator Oracle NLL result

Settings	Epoch	Discriminator Oracle NLL	
		Baseline	Enhanced
1	30	0.2480	0.5479
	50	0.2275	0.4585
2	30	0.3896	0.5628
	50	0.0277	0.5314

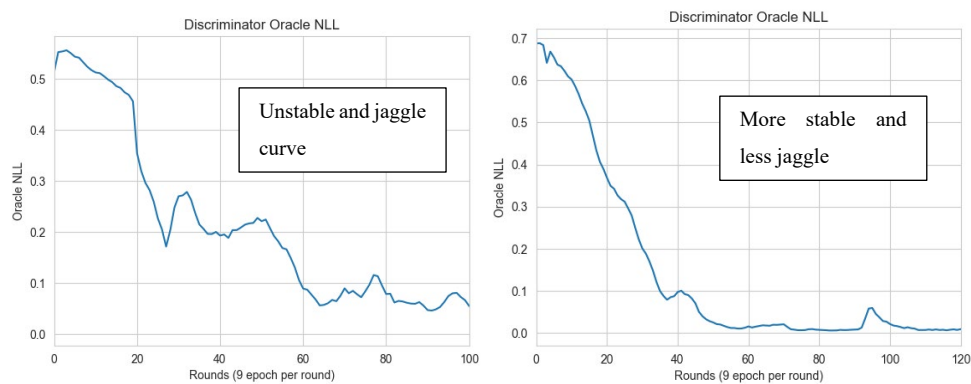


Figure 5.7: Discriminator Oracle NLL of baseline model using setting 1 and setting 2

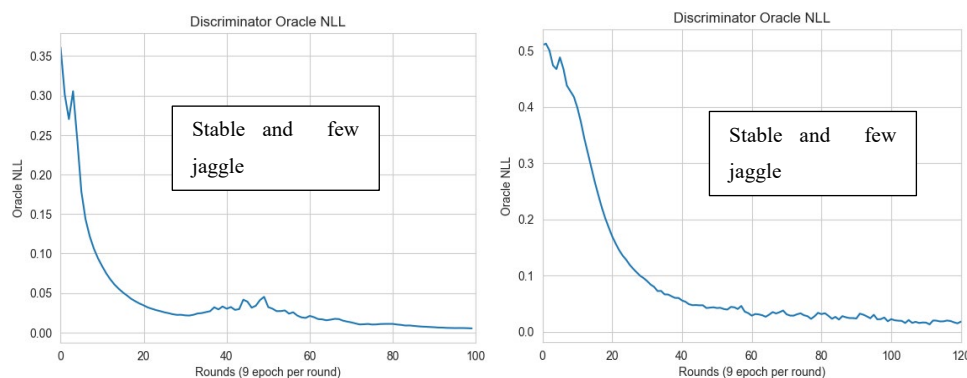


Figure 5.8 :Discriminator Oracle NLL of enhanced model using setting 1 and setting 2

In Setting 1, the baseline model decreased from 0.2480 to 0.2275, while the enhanced model decreased from 0.5479 to 0.4585. In Setting 2, the baseline model decreases from 0.3896 to 0.0277 while the enhanced model change from 0.5628 to 0.5314. The baseline model showed more improvement in Discriminator Oracle NLL, particularly in Setting 2. In Figure 5.7, the baseline discriminator Oracle NLL consists of many frustrations which indicates the instability of the model. This shows that the discriminators in the baseline model struggle to identify the differences between real and fake data. This has shown that the discriminator in the enhanced model performs more stable and is better at distinguishing real from generated data in some configurations.

The baseline model's fluctuations indicate its difficulty in consistently distinguishing between real and generated data, while the enhanced model shows a more stable performance. The lower fluctuations in the enhanced model suggest that the gradient penalty keeps the discriminator's updates gradual, ensuring it doesn't dominate the generator too early in the training process.

5.1.1.4 BLEU Score

BLEU score helps to evaluate the quality of the generated text sequences by the SeqGAN model. It measures how similar the generated poem is to a set of reference poems. The score helps measure the fluency of the generated text by checking if the generated words and phrases are in the correct order and appropriate for the context. Higher BLEU scores for the generator indicate better-quality generated text, as the generated sequences closely match the

reference text. It helps to observe how the architectural improvements impact the quality and coherence of the generated poem.

Table 5.6: BLEU Score results for both model

Epoch	BLEU score	
	Baseline Model	Enhanced Model
20	0.25	0.30
40	0.29	0.38
60	0.32	0.45
80	0.34	0.50
100	0.35	0.53

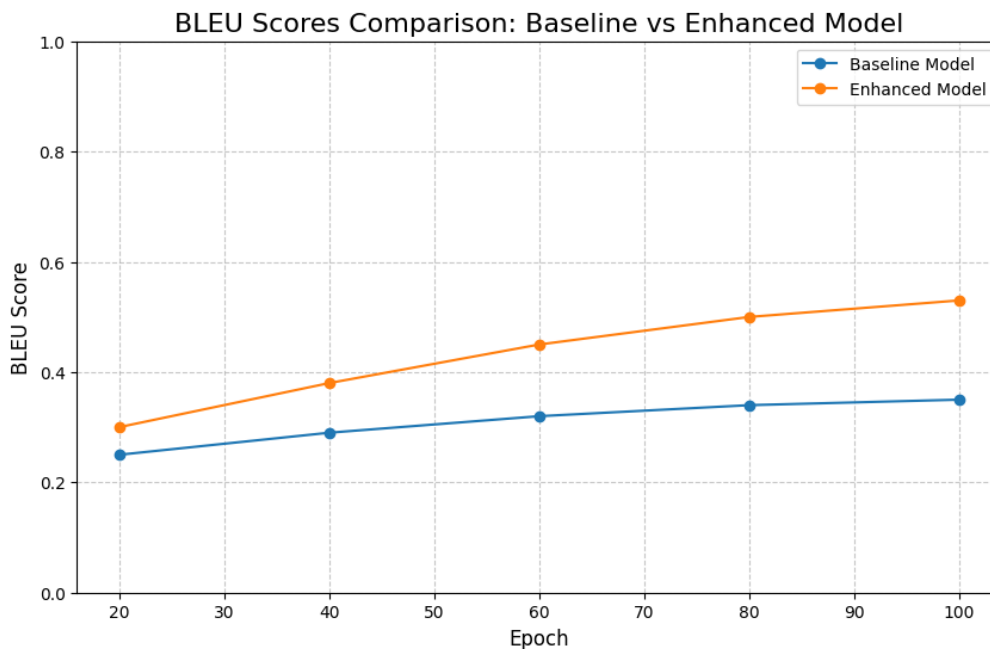


Figure 5.9: BLEU Scores Comparison between Baseline and Enhanced Model

The enhanced model shows a more rapid improvement in BLEU scores. From epoch 20 to 100, the enhanced model's BLEU score increased by 0.23 (from 0.30 to 0.53), while the baseline model only improved by 0.10 (from 0.25 to 0.35). Both models show a diminishing rate of improvement as training

progresses. However, the enhanced model maintains a steeper improvement curve throughout, suggesting it continues to learn more effectively from additional training data. The improved BLEU scores indicate that the architectural innovations integrated into the enhanced model, such as the GRU layers and attention mechanisms, provide a superior capacity to learn long-range dependencies and improve narrative coherence over time. These enhancements allow the generator to produce sequences that better mimic human-like coherence and structure, contributing to higher BLEU scores as training progresses.

Discussion

In Setting 1, the enhanced model's final generator loss (0.0186) was approximately 258 times lower than the baseline model's loss (4.7949) after 100 epochs. Similarly, in Setting 2, the enhanced model achieved a final loss of 0.131, about 36 times lower than the baseline's 4.7621 after 120 epochs. This reduction in generator loss indicates that the enhanced model is more effective at producing sequences that closely resemble the true data distribution. The practical implication is that the enhanced model is likely to generate higher quality, more coherent poems that better mimic human-written text.

The Oracle Negative Log-Likelihood (NLL) scores further corroborate this improvement. In Setting 1, the enhanced model's Oracle NLL decreased from 6.1021 to 2.8082 over 100 epochs, while the baseline model's score worsened slightly, increasing from 11.5810 to 11.7869. This divergence indicates that the enhanced model's generated sequences are progressively aligning closer with the true data distribution, while the baseline model struggles to make similar progress. The lower final Oracle NLL score of the enhanced model (2.8082 vs. 11.7869) quantifiably demonstrates its superior generation capabilities.

In addition to the generator and discriminator performance metrics, the BLEU (Bilingual Evaluation Understudy) scores provide crucial insight into the quality of the generated text. By epoch 100, the enhanced model achieves a BLEU score of 0.53, which is 51.4% higher than the baseline model's score of 0.35. This substantial difference indicates that the text generated by the enhanced model is significantly more similar to human-written reference texts. However, the enhanced model's BLEU score is still increasing at epoch 100 at a slower rate. This indicates that there is room for further improvement with extended training, while the baseline model appears to be stable.

Besides that, the discriminator performance presents a more significant result. In Setting 1, the enhanced model's discriminator showed a remarkable improvement with the loss decreasing from 0.5267 to 0.0318 over 50 epochs, compared to the baseline's improvement from 0.6770 to 0.6612. Conversely, in Setting 2, both models showed minimal change in discriminator loss, baseline: 0.5629 to 0.5435; enhanced: 0.5628 to 0.5314. This inconsistency across settings suggests that while the enhanced model's generator consistently outperforms the baseline, the discriminator's performance is more sensitive to hyperparameter configurations.

The enhanced model also demonstrates faster convergence by reaching lower loss values in fewer epochs. This is evident in the generator loss trajectories where the enhanced model shows a steeper decline compared to the baseline. For example, in Setting 1, the enhanced model's generator loss dropped by 99.3% (from 2.5078 to 0.0186) over 100 epochs, while the baseline model's loss only decreased by 16.5% (from 5.7422 to 4.7949) in the same period. This faster convergence translates to potential savings in computational resources and training time.

The sensitivity to hyperparameter changes is another important observation. The performance gap between Settings 1 and 2, especially in discriminator loss underscores the critical role of hyperparameter tuning in

GAN training. For example, the enhanced model's discriminator in Setting 1 achieved a final loss of 0.0318, while in Setting 2, it only reached 0.5314. This substantial difference highlights the need for careful optimization of hyperparameters to fully leverage the potential of the enhanced architecture.

In conclusion, the quantitative results support the enhanced SeqGAN model in terms of generation quality and training efficiency. The integration of GRU, Transformer architecture, and gradient penalty has led to substantial improvements in generator performance across multiple metrics. However, the mixed results in discriminator performance and the observed sensitivity to hyperparameters indicate areas for further research and optimisation. Future work should focus on refining the discriminator architecture, conducting more extensive hyperparameter searches, and evaluating the model's performance on diverse, real-world datasets to ensure that these improvements translate effectively to practical applications in poem generation.

5.1.2 Evaluation of Poem

The generated poem is evaluated based on their adherence to the traditional five-character quatrain (五言绝句) form, a classic style of Chinese poetry since the dataset poems that we used to train the generator are mostly in five-character quatrain form.

The five-character quatrain consists of four lines, each containing exactly five characters. This form is known for its strict structural rules, including specific requirements for parallelism and rhyme scheme. The poems were analysed based on the character count, line count, parallelism and rhyme scheme.

意邛志弃醉 杖吁如学姑 翳雕武纷天 须川驾惭尝
 近酒静水飒 织在影戏伤 怯草药若菲 客雪黄艳愁
 江峰落自曲 密茂重雾房 法岩正镜青 折对去者吾
 翊限人尘君 复笑谷车影 生凰殿烟敷 太肯日白得
 讲粉中尘虽 照艳日月本 哀辔亦若相 江堕依空苔
 峰钟路还猿 空稀应日柏 南应佛云急 舞度零歌花
 知音碎免濡 响丹破装倡 真佩职沼戟 佩合雾临冰
 天胡事扰轻 摇浣窗汉马 妆咏叶弦轻 关峰御飞凤
 望今息奈秀 晚楚滋暗悱 机结嶂早种 迹扶露鹳鸾
 炽絮境卿孙 洗宁说妍俱 疲寂鹤通含 抱鹤临渚盖
 北最观道与 安或还强失 恒歇琴阳年 累男日如弱
 镜时梅芳惜 影唐渔方胡 边承英君卿 嘉观早捧布
 思春水瓮前 汉駭臭徒叹 求暂闲蓬荡 陇头人君楚
 阳海边空 远自惜善今 寿风淹天今 寂此自昔相
 寒冻次梅度 辨齐铜壮昔 嶂退履交以 夷邪惆阳昔
 为树声响渭 对还叹仆犯 入分甲窄出 尊觉携友武
 迨欢兹阳众 连接管房抵 笔观可新厌 李妾日台有
 古四碣芳池 双闭啼池房 憎遮淹如胡 庭遇待霜郡
 惕寐兹戈军 拜盟皇龙化 巍髯积岳松 公申自善衡
 椒牛银笑懒 峨林疏悲笛 牵橘虚白条 达贫自田灭

Figure 5.11: Generated poems using
baseline model

Line Length and Structure

From Figure 5.10 and Figure 5.11, both generated poems fulfil the requirements of the character count, which is all lines contain exactly five characters. They also fulfilled the line count which consisted of four lines.

Parallelism and Rhythm

Parallelism in Chinese poetry refers to the use of similar grammatical structures or thematic elements across lines. In a **five-character quatrain** (五言绝句), this often means that the first and third lines (or the second and fourth lines) should reflect each other in terms of structure or meaning. This mirroring effect creates a sense of balance and harmony within the poem, allowing the poet to explore related themes or ideas in parallel lines. Additionally, the repetition of similar structures enhances the musicality and flow of the poem, making it more aesthetically pleasing.

腐在霜书骑 回道光行鸟 有尘亦冀裾 遵酌棹雪四
 穴咽洞肩旷 低芊蛭重年 胡边关亭干 编拱琐蕙屣
 摘豁耿承有 夕惊晚乐德 度中塞侍琴 摇草愁与乘
 申响报意搥 俊毁鞞新吹 空树出来万 衣已琴在结
 茶膏蚀薄勒 枫昨祈元依 属俸蚊逃躅 岩尘有景地
 巢隳萃鬣簟 计契辱瑟奈 熟扰创斤摘 鹧孜进循被
 胞绮约翎掠 短矶床尝尼 痒更时气隘 是别今沙晚
 蚊舂棱赠避 卯虬匡揭迤 缚招浆仲弓 匠葛鬻牙鲋
 廖楷堂我手 知王喜霏浦 縠霏敲颞挫 铜豫渺拟明
 橐墩井英偈 口朔为在四 酒已光顶泚 独阴阴列气
 蕤鬣纒纒珂 繡绀今远苑 春欲杯兰榼 植亡镜洪变
 鸭荣锄沿驻 弘门咸全眺 息流都首远 白雪重离情
 榛仁第官魅 辜障没树一 葶殒亡谁驱 衢觞寥隼坤
 泳钝鸩算醴 雒专瀑殷鹭 藟洸兴知安 无望初远尽
 虞娇渊虔陆 颖扶比纳榼 阙干更藤嵯 咿旒旋山讳
 茯照摇剑庭 立摆髡琪据 邓窠暝予芭 切祀嘶鸣淬
 婧啸灌龙行 夕动干惨李 夕还兢旱遽 菲隳钱莞雅
 珉肘渚泾饥 鹭鬻柯浣獬 剧斤旂炆昆 访弊喘甬葱
 鸱梵熬摇成 持红庭腴峦 吕兆数倦阔 燕中然望关
 沔乳佺屹甯 肩苛低纒输 杼汲趟颞范 袒唏诀罍踞

Figure 5.10: Generated poems using
enhanced model

In traditional Chinese poetry, a rhyme scheme typically follows the ABAB pattern. This means that the first and third lines rhyme with each other, while the second and fourth lines rhyme with each other.

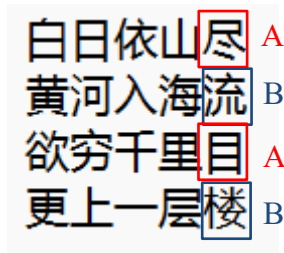


Figure 5.12: Climbing White Stork Tower from Wang Zhihuan

Figure 5.12 shows the quatrain the first line ends with 尽 (jìn) and the third line ends with 目 (mù), which rhyme with each other, forming the "A" rhyme. Similarly, the second line ends with 流 (liú) and the fourth line ends with 楼 (lóu), forming the "B" rhyme. This ABAB rhyme scheme provides a structured and melodic flow to the poem.

The parallelism in this example can also be seen in the themes of the lines. The first and third lines both focus on expanding sight or perspective, while the second and fourth lines emphasize movement and progression. This thematic parallelism reinforces the meaning of the poem, while the ABAB rhyme scheme enhances its rhythm and aesthetic quality.

Sample Lines of the Baseline Model

意邛志弃醉	First line
杖吁如学姑	Second line
翳雕武纷天	Third line
须川驾惭尝	Forth line

Figure 5.13: Sample lines poems from the baseline model

Figure 5.13 shows a sample line poem generated by the baseline model. The first line of the poem describes a scene but it lacks clear grammatical and thematic parallelism with the other lines. The second line seems disconnected

in terms of thematic content and structure compared to the other lines. The third line introduces a different theme and does not align well with the previous lines. While the last line presents yet another thematic element, and it does not mirror the structure or content of the other lines.

As for the rhyme of the poem, the first and third lines of the poem do not rhyme with each other or with the other lines. The second and fourth lines also do not exhibit a clear rhyme scheme with each other or with the first pair of lines. This line of poems lacks a distinct rhyme scheme. The lines do not follow the traditional ABAB rhyme pattern, making it less aligned with classical five-character quatrains.

Sample Lines of Enhanced Model

腐在霜书骑	First line
回道光行鸟	Second line
有尘亦箕裾	Third line
遵酌棹雪四	Forth line

Figure 5.14: Sample lines poems for enhanced model

Figure 5.14 shows a sample line poem generated by an enhanced model (GRU + Transformer). The first line of the poem introduces imagery related to frost and decay, which can be seen as setting a scene. The second line continues the scene setting with imagery of light and birds, showing a thematic connection to the first line. The third line introduces dust and a garment, which maintains thematic continuity with the previous lines. The last line describes actions and snow, connecting to the overall imagery of the poem. This generated poem shows more consistent thematic parallelism. The lines relate to each other through imagery and description, creating a coherent theme.

As for the rhyme of the poem, the first and third lines of the poem may rhyme, particularly in the final characters “骑” (qi) and “裾” (xu) depending on pronunciation. The second and fourth lines also might rhyme, with “鸟” (niao) and “四” (si) potentially forming a rhyme pair. The overall poem has a more recognizable rhyme scheme, possibly following an ABAB pattern which aligns better with traditional quatrain standards.

5.2 System Demonstrations

5.2.1 Model Enhancement

5.2.1.1 GRU Implementation

GRU (Gated Recurrent Unit) is a type of RNN architecture that is often used for handling sequential data.

```
class TargetGRU(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, use_cuda):
        """
        Initializes the GRU-based generator model.

        Args:
            vocab_size (int): Size of the vocabulary (number of unique tokens).
            embedding_dim (int): Dimension of word embeddings.
            hidden_dim (int): Dimension of the hidden state of the GRU.
            use_cuda (bool): Whether to use CUDA (GPU) for computations.
        """
        super(TargetGRU, self).__init__()
        self.hidden_dim = hidden_dim
        self.use_cuda = use_cuda

        # Embedding layer maps input tokens (integers) to dense vectors of size embedding_dim
        self.embed = nn.Embedding(vocab_size, embedding_dim)

        # GRU layer to process sequential data. Takes embeddings as input and outputs hidden states.
        # batch_first=True means input and output tensors will have the shape (batch_size, seq_len, feature_size)
        self.gru = nn.GRU(embedding_dim, hidden_dim, batch_first=True)

        # Fully connected layer that projects GRU outputs to the size of the vocabulary.
        self.fc = nn.Linear(hidden_dim, vocab_size)

        # Log softmax function is applied to the output to convert the logits to log probabilities.
        self.log_softmax = nn.LogSoftmax(dim=-1)
```

Figure 5.15: Overview of TargetGRU class

GRU is implemented in the TargetGRU class and serves as a target model to replace LSTM for evaluation. It processes the input sequence and captures temporal dependencies. The key components of GRU are a hidden state that carries information forward as the model processes the sequences; an update gate to determine how much of the previous hidden state should be carried

forward; and a reset gate to control how much of the previous hidden state should be ignored.

```
self.embed = nn.Embedding(vocab_size, embedding_dim)
```

Figure 5.16: Embedding Layer

An embedding layer is created to convert input tokens which are represented as integers, into dense vectors of fixed-size `embedding_dim`.

```
self.gru = nn.GRU(embedding_dim, hidden_dim, batch_first=True)
```

Figure 5.17: GRU Layer

A gated recurrent unit that processes sequences and maintains a hidden state across timesteps. It uses the token embeddings as input and produces hidden states that capture the sequence's context. It takes the input size `embedding_dim` and produces the output of size `hidden_dim`. The `'batch_first=True'` means the input and output tensors are provided as (batch, seq, feature).

```
self.fc = nn.Linear(hidden_dim, vocab_size)
self.log_softmax = nn.LogSoftmax(dim=-1)
```

Figure 5.18: Fully Connected Layer and Softmax

A fully connected layer is created to map the GRU output back to vocabulary size so that the model can predict the next token. The log softmax will then convert the raw output from the fully connected layer into log probabilities.

```
def forward(self, x):
    embeds = self.embed(x)
    gru_out, _ = self.gru(embeds)
```

Figure 5.19: Forward Pass

The forward function is created to pass the network. It first embeds the input tokens by converting them into embedding vectors and then processes the embedded sequence through the GRU layer. The hidden state is also being updated at each timestep.

```
fc_out = self.fc(gru_out)
return self.log_softmax(fc_out)
```

Figure 5.20: Overview of TargetGRU class

After that, the GRU output is passed through the fully connected layer and then the log softmax converts the output to log probabilities, which represent the likelihood of each token in the vocabulary being the next token.

```
def step(self, x, h):
    """
    Single step forward pass through the GRU, generating one token at a time.

    Args:
        x (torch.Tensor): Input tensor of shape (batch_size, 1), containing one token per batch.
        h (torch.Tensor): Hidden state from the previous time step (1, batch_size, hidden_dim).

    Returns:
        torch.Tensor: Log probabilities for the next token (batch_size, vocab_size).
        torch.Tensor: Updated hidden state after processing this token.
    """
    # Flatten GRU parameters for better GPU utilization (if necessary).
    self.gru.flatten_parameters()

    # Step 1: Embed the current token.
    emb = self.embed(x) # Shape: (batch_size, 1, embedding_dim)

    # Step 2: Process the embedding through the GRU to update the hidden state.
    out, h = self.gru(emb, h) # out: (batch_size, 1, hidden_dim), h: updated hidden state

    # Step 3: Project the GRU output to vocabulary size using the fully connected layer.
    # Reshape the output to (batch_size, hidden_dim) before passing to fc.
    out = self.log_softmax(self.fc(out.contiguous().view(-1, self.hidden_dim))) # Shape: (batch_size, vocab_size)

    # Return the log probabilities for the next token and the updated hidden state.
    return out, h
```

Figure 5.21: Step Method for Token-by-Token Processing

The ‘step’ method handles generating one token at a time, useful in sampling where the model needs to generate tokens sequentially. First, the token is converted into its embedding vector. After that the embedding is processed by GRU and the hidden state is updated based on the token. The fully connected

layers then map the GRU output to vocabulary size. Finally, the LogSoftmax converts the output to log probabilities, which represent the likelihood of each token in the vocabulary being the next token.

```
def init_hidden(self, batch_size):
    """
    Initializes the hidden state for the GRU. This is required at the start of a new sequence.

    Args:
        batch_size (int): Number of sequences in the batch.
    """
    # Initialize hidden state with zeros. Shape: (1, batch_size, hidden_dim)
    h = torch.zeros((1, batch_size, self.hidden_dim))

    # Move the hidden state to the GPU if CUDA is enabled.
    if self.use_cuda:
        h = h.cuda()
    return h
```

Figure 5.22: `init_hidden` function

Initializes the GRU's hidden state at the start of generating a new sequence. The hidden state is filled with zeros and moved to the GPU if CUDA is enabled.

```
def init_params(self):
    """
    Initializes model parameters (weights and biases) with random values.
    """
    # For each parameter in the model, initialize its data with a normal distribution.
    for param in self.parameters():
        param.data.normal_(mean=0, std=1)
```

Figure 5.23: `init_params` function

This `init_params` function will randomly initialize the model's weights and biases using a normal distribution. This can help improve training stability.

5.2.1.2 Transformer

The Transformer architecture is an alternative to RNNs and GRUs, utilizing self-attention mechanisms to handle sequences more efficiently.

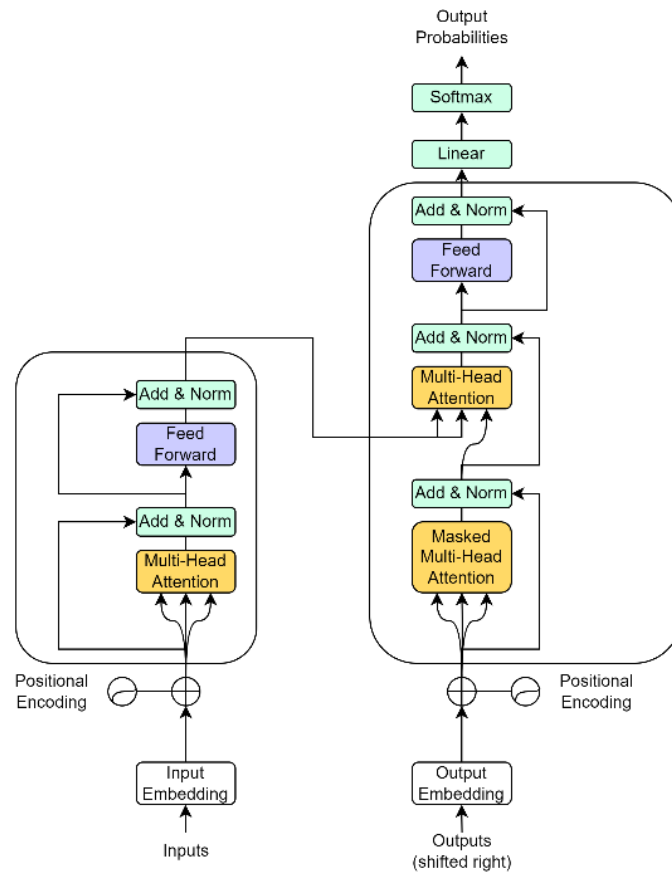


Figure 5.24: Architecture of Transformer

Figure 5.24 shows an Transformer architecture. Transformers are known for their parallelization and ability to capture long-range dependencies across sequences. It consists of two key components which are self-attention and positional encoding. The self-attention component helps the model focus on relevant parts of the sequence when generating the next word, while the positional encoding is used to maintain the order since transformers do not.


```

class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        """
        Implements positional encoding as described in the 'Attention is All You Need' paper.

        Args:
            d_model (int): The dimension of the embeddings (model size).
            dropout (float): Dropout rate applied after adding positional encoding.
            max_len (int): The maximum length of sequences for which to precompute positional encodings.
        """
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout) # Dropout layer for regularization.

        # Positional encoding matrix initialized with zeros.
        pe = torch.zeros(max_len, d_model)

        # Compute the position indices, from 0 to max_len-1.
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)

        # Compute the division term used in the sine and cosine functions.
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))

        # Apply sine to even indices in the positional encoding.
        pe[:, 0::2] = torch.sin(position * div_term)

        # Apply cosine to odd indices in the positional encoding.
        pe[:, 1::2] = torch.cos(position * div_term)

        # Add an extra dimension to `pe` so that it can be added to the input embeddings.
        pe = pe.unsqueeze(0)
        self.register_buffer(name='pe', pe)

```

Figure 5.25: PositionEncoding method

The figure 5.25 shows an overall implementation of the PositionEncoding method that is used to add positional to word embeddings in transformer models. The transformer architecture is unlike RNNs or GRUs. Because it does not record sequential order by default, positional encoding gives the model information about where each word in the sequence is located. It helps the model keep track of the positions of tokens within a sequence which allows it to capture order and relationships between tokens.

```

# Positional encoding matrix initialized with zeros.
pe = torch.zeros(max_len, d_model)

```

Figure 5.26: Positional Encoding Matrix

A matrix `pe` is created to hold the positional encodings for all positions up to `max_len` for which to precompute positional encodings. The size of this matrix is $[\text{max_len}, \text{d_model}]$, where each row represents the positional encoding for a specific position in the sequence. `max_len` is the maximum length of any

sequence that will be processed by the model, and ‘d_model’ is the size of the embedding dimension.

```
# Compute the division term used in the sine and cosine functions.
div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))
```

Figure 5.27: Division Term

The ‘div_term’ is used to scale the position indices by different frequencies for the sine and cosine functions. This ensures that the model can distinguish positions based on different periodic functions. The logarithmic scaling ($-\log(10000) / d_model$) ensures that the positional encodings vary smoothly across the sequence.

```
# Apply sine to even indices and cosine to odd indices in the positional encoding.
pe[:, 0::2] = torch.sin(position * div_term)
pe[:, 1::2] = torch.cos(position * div_term)
```

Figure 5.28: Sine and Cosine Functions

The sine function is applied to even indices of the embedding dimension, and the cosine function is applied to odd indices. These alternating sine and cosine functions help the model learn the positional relationships between tokens.

```
# Add an extra dimension to 'pe' so that it can be added to the input embeddings.
pe = pe.unsqueeze(0)
self.register_buffer(name='pe', pe)
```

Figure 5.29: Unsqueeze and Register Buffer

The ‘pe’ matrix is reshaped using ‘unsqueeze(0)’ to add a batch dimension, resulting in shape [1, max_len, d_model]. This allows it to be broadcast across different batches of input sequences. On the other hand, ‘self.register_buffer(‘pe’, pe)’ ensures that ‘pe’ is stored as part of the model, but it is not a learnable parameter. This means it won't be updated during training via backpropagation, but it is persistent and saved with the model.

```

def forward(self, x):
    seq_len = x.size(1)
    if seq_len <= self.max_len:
        return x + self.pe[:, :seq_len, :]
    else:
        # For sequences longer than max_len, repeat the positional encodings as needed
        return x + self.pe[:, :self.max_len, :].repeat(1, math.ceil(seq_len / self.max_len), 1)[:, :seq_len, :]

```

Figure 5.30: Forward Pass Function

In the forward pass, the input tensor x (the embedded sequence) has the precomputed positional encodings added to it. If the sequence length is within the precomputed ‘max_len’, the positional encodings are added directly to the input embeddings. However, if the sequence length exceeds ‘max_len’, the positional encoding matrix is repeated using `repeat()`. The result is sliced to fit the exact ‘seq_len’ which allows the model to handle longer sequences without breaking. This ensures that even if the sequence length exceeds the precomputed maximum length, the model can continue cyclically adding positional encodings.

5.2.1.3 Gradient Penalty

Gradient Penalty is integrated to enforce Lipschitz continuity, which ensures more stable training by penalizing the model when the gradient norm moves away from 1. This regularization helps prevent mode collapse, which is important when generating diverse poems. A penalty term is added to the loss function, which pushes the gradient norm towards 1. This helps to stabilize the discriminator's gradients and stay well-behaved to obtain better training dynamics for both the generator and discriminator.

```

def compute_gradient_penalty(D, real_samples, fake_samples):
    # Generate random weights for interpolation between real and fake samples.
    alpha = torch.rand(real_samples.size(0), 1, 1, 1).to(real_samples.device)

    # Create interpolated samples by mixing real and fake samples.
    interpolates = (alpha * real_samples + ((1 - alpha) * fake_samples)).requires_grad_(True)

    # Pass the interpolated samples through the discriminator to get predictions.
    d_interpolates = D(interpolates)

    # Create a tensor of ones, which will be used to calculate the gradients.
    fake = torch.ones(d_interpolates.size()).to(real_samples.device)

    # Compute the gradients of D(interpolates) with respect to the interpolated samples.
    gradients = torch.autograd.grad(
        outputs=d_interpolates,
        inputs=interpolates,
        grad_outputs=fake,
        create_graph=True,
        retain_graph=True,
        only_inputs=True
    )[0]

    # Reshape the gradients to a 2D tensor where each row corresponds to a single sample.
    gradients = gradients.view(gradients.size(0), -1)

    # Compute the norm of the gradients for each sample.
    gradient_norm = gradients.norm(p=2, dim=1)

    # Compute the gradient penalty by measuring how much the gradient norm deviates from 1.
    gradient_penalty = ((gradient_norm - 1) ** 2).mean()
    return gradient_penalty

```

Figure 5.31 : Function of Gradient Penalty Calculation

The figure shows an overall function of ‘compute_gradient_penalty’ that is used to calculate the gradient penalty in the context of SeqGAN. This function enforces the Lipschitz constraint by penalizing the norm of the gradients of the discriminator output with respect to interpolated samples. Two arguments are used in this function, which is a batch of real data samples and a batch of generated data samples generated from the generator. This function will return a scalar value that is added to the discriminator loss, which is the gradient penalty. The function will be called during the discriminator adversarial training.

```

# Generate random weights for interpolation between real and fake samples.
alpha = torch.rand(real_samples.size(0), 1, 1, 1).to(real_samples.device)

# Create interpolated samples by mixing real and fake samples.
interpolates = (alpha * real_samples + ((1 - alpha) * fake_samples)).requires_grad_(True)

```

Figure 5.32: Interpolation Between Real and Fake Samples

A random alpha is generated for each sample in the batch, and it is used to create an interpolated sample by mixing real and fake samples. These interpolated samples help the GAN enforce the Lipschitz continuity condition, such as keeping the gradients bounded.

```
# Pass the interpolated samples through the discriminator to get predictions.
d_interpolates = D(interpolates)
```

Figure 5.33: Discriminator Output on Interpolated Samples

The interpolated samples are passed through the discriminator D, which gives the predictions `d_interpolates`. These predictions are used to compute the gradients for the interpolated inputs.

```
# Compute the gradients of D(interpolates) with respect to the interpolated samples.
gradients = torch.autograd.grad(
    outputs=d_interpolates,
    inputs=interpolates,
    grad_outputs=fake,
    create_graph=True,
    retain_graph=True,
    only_inputs=True
)[0]
```

Figure 5.34: Gradient Computation

‘`torch.autograd.grad`’ is used to compute the gradients of `D(interpolates)` for the interpolated samples. The ‘`grad_outputs=fake`’ argument ensures that the gradient is calculated in the right direction as a backward pass. The ‘`create_graph=True`’ argument is to compute the gradient penalty that requires the computation graph to remain intact for higher-order derivatives.

```
# Reshape the gradients to a 2D tensor where each row corresponds to a single sample.
gradients = gradients.view(gradients.size(0), -1)

# Compute the norm of the gradients for each sample.
gradient_norm = gradients.norm(p=2, dim=1)
```

Figure 5.35: Reshape Gradients and Gradient Norm

After that, the gradients are reshaped into a 2D tensor of shape (batch_size, num_features), where each row represents the gradient for a single sample. The L2 norm (Euclidean norm) of the gradients is computed for each sample. The Lipschitz constraint requires the norm of the gradients to be approximately 1.

```
# Compute the gradient penalty by measuring how much the gradient norm deviates from 1.
gradient_penalty = ((gradient_norm - 1) ** 2).mean()
return gradient_penalty
```

Figure 5.36: Gradient Penalty calculation formation

The penalty is calculated as the squared difference between the gradient norm and 1. It enforces that the gradients should remain close to 1 to ensure smoothness in the discriminator's behaviour. The penalty is averaged across the batch and returned as a scalar. This value is then added to the discriminator's loss to penalize large deviations from the Lipschitz constraint.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusion

This research successfully enhanced SeqGAN for poem generation by integrating GRU, Transformer architecture, and gradient penalty which resulted in significant improvements in text quality, learning efficiency, and model stability. The enhanced model consistently outperformed the baseline across all evaluation metrics, with a notable 51.4% increase in BLEU scores, demonstrating a marked improvement in the similarity between generated and human-written poems. Additionally, the enhanced model achieved faster convergence, with the generator loss decreasing significantly compared to the baseline, showing a more efficient learning process. The gradient penalty also contributed to more stable training across different hyperparameter settings. Evaluation metrics, including generator and discriminator loss, Oracle Negative Log-Likelihood (NLL), and BLEU scores, provided a comprehensive assessment of both the technical and practical performance of the models. Overall, this research demonstrates that architectural improvements can significantly enhance SeqGAN's ability to generate high-quality, coherent poetry, paving the way for more advanced text generation systems.

6.2 Recommendations for future work

Extended Training and Larger Datasets

The enhanced model showed signs of continued improvement even at later epochs. Experimenting with longer training periods (beyond 100 epochs) and larger, more diverse poem datasets such as including poems in different languages, styles, and structures, could potentially lead to even higher BLEU scores and more versatile poem generation. Larger datasets could be sourced from publicly available poetry corpora, or generated synthetically to diversify the content. Training time would need to be carefully monitored to prevent overfitting, especially as the dataset size increases. Implementing early stopping mechanisms or checkpoints can help mitigate this risk. Larger datasets are

essential to improving the versatility and creative range of the model. Extended training ensures the model can explore deeper into its optimization landscape, potentially uncovering patterns that shorter training cycles might miss.

Human Evaluation Integration

BLEU scores provide a good proxy for text quality but they don't capture all aspects of poetic merit. Implementing an evaluation phase into the reward structure may help refine the model's ability to produce text that aligns with subjective quality standards and provides insights into aspects like creativity, emotional impact, and adherence to poetic forms that automated metrics might miss.

A human evaluation framework could be introduced by crowd-sourcing ratings on dimensions such as creativity, emotional resonance, and adherence to structure from professional poets or online users. These ratings can be fed into the model as rewards, utilizing reinforcement learning techniques to further fine-tune the generator's performance based on human feedback. However, integrating human evaluations introduces subjectivity into the training process, which could be inconsistent and difficult to quantify. Gathering reliable and diverse feedback at scale is time-consuming and costly. Furthermore, balancing between automated BLEU scores and human assessments is complex, as the model may have to learn to optimize for conflicting objectives.

Hyperparameter Tuning

The performance of the model showed sensitivity to hyperparameter changes, particularly in discriminator training. Conducting a more extensive hyperparameter search using techniques like Bayesian optimization or genetic algorithms could help to identify optimal configurations for different use cases or dataset sizes.

Hyperparameter tuning could be automated through techniques such as Bayesian optimization, iteratively selects hyperparameters that minimize a given loss function, or genetic algorithms that evolve hyperparameters over generations. Tools like Optuna or Hyperopt can be integrated into the model's

training pipeline to automate this process. Hyperparameters such as learning rates, batch size, and discriminator-pretraining steps can be tuned to optimize model performance for specific datasets and tasks. However, the increased complexity in the model tuning process may lead to longer training times and the need for more computational resources. Additionally, over-optimization of hyperparameters for a specific dataset might result in a model that does not generalize well to new or unseen data, requiring a balance between exploration and exploitation in the tuning process.

REFERENCES

- Chen, M. et al., 2018. Top-K Off-Policy Correction for a REINFORCE Recommender System. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*.
- Che, T. et al., 2017. Maximum-Likelihood Augmented Discrete Generative Adversarial Networks.
- Dai, W. et al., 2018. Toward Understanding the Impact of Staleness in Distributed Machine Learning.
- Duarte, F. F., Lau, N., Pereira, A. & Reis, L. P., 2020. A Survey of Planning and Learning in Games. *Applied Sciences*, 10(13), p. 4529.
- Eck, D. & Schmidhuber, J., 2002. A First Look at Music Composition using LSTM Recurrent Neural Networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*.
- Espeholt, L. et al., 2018. IMPALA: Scalable distributed deep-RL with importance-weighted actor-learner architectures. *In International Conference on Machine Learning*, pp. 1406-1415.
- Goodfellow, I. et al., 2014. Generative adversarial nets. *In Advances in neural information processing system*, pp. 2672-2680.
- Graves, A., Mohamed, A.-r. & Hinton, G., 2013. Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645-6649.
- Hochreiter, S. & Schmidhuber, J., 1997. Long short-term memory. *Neural Computation*, 9(8), pp. 1735-1780.
- Holtzman, A. et al., 2019. The Curious Case of Neural Text Degeneration.. *In International Conference on Learning Representations*.
- Krivosheev, N., Vik, K., Ivanova, Y. & Spitsyn, V., 2021. Investigation of the Batch Size Influence on the Quality of Text Generation by the SeqGAN Neural Network. *GraphiCon*, Volume 3027, pp. 1005-1010.
- Lagutin, E., Gavrilov, D. & Kalaidin, P., 2021. Implicit Unlikelihood Training: Improving Neural Text Generation with Reinforcement Learning.. *EACL*, pp. 1432-1441.

- Liu, X., Gao, J., Zhang, W. & Shou, L., 2020. Towards Robust Neural Machine Translation. *In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1777-1789.
- Martin Arjovsky, Chintala, S. & Bottou, L., 2017. Wasserstein GAN.
- Schulman, J. et al., 2015. Trust Region Policy Optimization. *Machine Learning*, p. 15.
- Schulman, J. et al., 2017. Proximal Policy Optimization Algorithms. *Machine Learning*.
- Sutskever, I., Vinyal, O. & Q. V. L., 2014. Sequence to Sequence Learning with Neural Networks.. *Neural Information Processing Systems*.
- Sutton, R. & Barto, A., 2018. *Reinforcement learning: An introduction*.. s.l.:MIT press.
- Sutton, R. S. & Barto, A. G., 2018. *Reinforcement learning: An introduction*. s.l.:s.n.
- Vaswani, A. et al., 2017. Attention is all you need.
- Vo, T., 2022. A Novel Semantic-Enhanced Text Graph Representation Learning Approach through Transformer Paradigm.. *Cybernetics and Systems*, 54(4), pp. 499-525.
- Williams, R. J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, Volume 8, pp. 229-256.
- Yu, L., Zhang, W., Wang, J. & Yu, Y., 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *In Thirty-First AAAI Conference on Artificial Intelligence*..
- Zhang, H. et al., 2022. Text Feature Adversarial Learning for Text Generation With Knowledge Transfer From GPT2.. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1-12.
- Zhang, Y. et al., 2017. Adversarial Feature Matching for Text Generation.

