Faculty of Information Technology
Monash University

Deep Sequence Models: Learning to Generate Data and Adversarial Attacks

A thesis submitted for the degree of Doctor of Philosophy at Monash University, Faculty of Information Technology

By
Mahmoud Ahmed Hossameldeen Mohammad

Supervisors: Prof. Dinh Phung
Dr. Trung Le
Dr. Viet Huynh
Dr. He Zhao

2021

Copyright Notice

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature:

Print Name: Mahmoud Ahmed Hossameldeen Mohammad Ahmed.

Date: 15 February 2021.

Publications

- Hossam, M., Le, T., Papasimeon, M., Huynh, V. and Phung, D., "Text Generation with Deep Variational GAN", *in the third workshop on Bayesian Deep Learning of the annual conference on Neural Information Processing Systems* (*NeurIPS)*, Montréal, Canada, 2018.

- Hossam, M., Le, T., Papasimeon, M., Huynh, V. and Phung, D., "OptiGAN: Generative Adversarial Networks for Goal Optimized Sequence Generation", *in the Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN),* Glasgow, UK, 2020.

- Hossam, M., Le, T., Zhao, E. and Phung, D., "Explain2Attack: Text Adversarial Attacks via Cross-Domain Interpretability", *in the Proceedings of the 25th International Conference on Pattern Recognition, (ICPR)*, Milan, Italy, 2020.

- Hossam, M., Le, T., Zhao, E., Huynh, V. and Phung, D., "Improved and Efficient Text Adversarial Attacks using Target Information", *in the Robust and Reliable Machine Learning in the Real World Workshop (RobustML), International Conference on Learning Representations (ICLR)*, 2021.

Abstract

Teaching machines to process and understand sequence data is a critical component for advanced artificial intelligence (AI) systems. In many AI applications, including natural language understanding, drug discovery, and autonomous vehicles, sequences of various data types are used to learn and execute certain tasks. There has been extensive progress towards building machine learning models for processing sequence data. However, most of these models were mainly used for classification or prediction tasks. Until recently, sequence models lacked the ability to become creative, or to accurately learn the statistical distribution of the given data and generate new data following this distribution. They also lacked a common sense understanding of the sequences, like natural language, where malicious inputs could easily confuse a highly accurate classification model.

With the advent of deep learning as the dominant approach for efficient learning from large amounts of data, there has been a growing interest into building sophisticated deep models to process and understand sequences. Nonetheless, employing deep learning to build AI systems for previously described applications faces two difficulties: *sequence generation*, and *adversarial attacks* on sequence models, which are the focus of this thesis. Recently, there have been increasing efforts to build deep models for sequence generation following the success of Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) in computer vision tasks. However, applying these models for sequence generation faced several difficulties, where the outputs are mainly uncontrolled and randomized, and there is a lack of output diversity compared to the original data. Specifically, there are two main challenges that emerge in generation tasks of deep sequential models; (1) the difficulty to capture various modes of underlying data distribution, known as "mode-collapse", which leads to low diversity outputs generated only from few learned modes, and (2) the lack of mechanisms to control generated outputs, hindering the controlled generation of sequences where the user can control the model's output to obtain data with desired properties or attributes. In addition, despite the success of deep learning in terms of accuracy metrics, research has shown that common deep learning models can be easily fooled by adversarial examples, which are malicious inputs that look like the training data, but slightly perturbed, in a way imperceptible to humans. A deep model could be attacked with such adversarial examples models, potentially causing significant impairment to the model's performance when deployed in the real-world. On the other side, Generating such adversarial examples can be used to improve the models robustness,

by augmenting the training set with the generated adversarial examples. Nonetheless, generating adversarial examples for sequences like natural language suffers from high computational complexity and number of accesses needed to the attacked model, known as the number of queries. For an adversarial attacking agent, reduced number of queries to the target model is desired to avoid suspicion towards its activity. This thesis aims to address these three challenges in deep sequence models: generative models *diversity*, *controlled generation*, and generating *adversarial attacks*.

First, we discuss our work on the mode-collapsing problem of sequence GANs, and present our framework that employs both maximum likelihood estimation with standard GAN objective to overcome the mode collapsing issue. We prove that our framework approximates the true underlying data distribution, and apply the framework to natural language generation task, where we empirically show that it achieves higher diversity scores compared to the baseline.

Second, we discuss our work on controlled sequence generation, named OptiGAN, where we extend our previously proposed framework by incorporating rewards signal using reinforcement learning to maximise desired domain rewards. OptiGAN is the first GAN-based generative model for sequences that addresses the diversity issue in a principled approach. We apply OptiGAN to two discrete and real-valued data generation tasks; natural language and air-combat trajectory generation. We show that our model outperformed the baseline with up to 20% improvement in the quality score, and up to 12% improvement in the diversity score. We thus show that our model is able to achieve better desired scores than standard GAN and Reinforcement Learning (RL) baselines, while not sacrificing output diversity, illustrating the potential usefulness of our framework in advanced controlled generation tasks.

Finally, we propose a novel framework, Explain2Attack, as an efficient framework to generate adversarial examples for natural language models. Explain2Attack employs interpretable models across different domains to efficiently learn how to attack the target model. Our method achieves or out-performs state-of-the-art methods in adversarial attack rates, yet with reduced number of queries. With a set of experiments on different natural language sentiment classification models, our method achieves up to 11.9% reduction in number of queries. Additionally, we show that attack rates and number of queries can further be improved by utilizing the target model information.

This thesis contributes novel models and algorithms to key challenges and advances knowledge in deep sequence modeling, which broadly impacts many important real-world applications.

Acknowledgments

Throughout my studies and the writing of this dissertation I have received great support and assistance. I first would like to express my gratitude to my supervisors, Professor Dinh Phung and Dr Trung Le for their continuous support and guidance. They have taught me many things, not only the scientific knowledge, tireless diligence, and initiative conduct, but also valuable personal and professional skills, humbleness, and a wellbeing approach to life.

I would like to specially thank Dr Viet Hynh and Dr He Zhao, for their mentorship and consistent availability for brainstorming. They always provided me with valuable feedback, insightful discussions and suggestions, and academic writing guidance, that were of great help to my research.

I am also grateful to Associate Professor Guido Tack, Dr Lan Du, and Dr Christoph Bergmeir for being on my progress reviews panel and offering support and helpful feedback for my research progress. I would like to extend my gratitude to Dr Michael Papasimeon and Defence Science and Technology Group, who financially supported part of my PhD study.

I am deeply grateful to my dear friends Xuanli He and Srinibas Swain, with whom I shared my daily life for the last three years, and through their brotherly friendship and technical help, surrounded me with a delightful and caring environment. I am also very thankful to my friends Laksri Wijerathna, Bhagya Hettige, Ruangsak Trakunphutthirak and Mostafa Rizk, who were a close family to me during my studies, and to Harald Bögeholz for the inspiring mathematical and technical discussions. I would like to thank my dear colleagues Nhan Dam, Van Nguyen, Dai Nguyen, and Hung Vu, for the insightful discussions, support and care. I am gratefully indebted to my close friends, Mohammad Ammar and Dr Wael Awad, who have been wholeheartedly supporting me in every possible way.

Transferring from Deakin University to Monash University, I am grateful to all the academics, staff members as well as PhD students at Centre for Pattern Recognition and Data Analytics (PRADA, now A2I2). I am deeply thankful to my previous supervisor, Dr Wei Luo, who through his care and support, helped me to overcome the knowledge gap needed for my studies. I am immensely grateful to Associate Professor Mohamed Abdelrazek, who greatly supported me since the first day of my PhD and throughout, with valuable advice and personal support at critical times. I am specially thankful to my friends Dang Nguyen and Majid Abdolshah, for their support and heartwarming discussions. I am also thankful to all professors at PRADA, and to Tinu Joy and Trang Pham, for their support during my last period at the lab.

Finally, I cannot express enough gratitude to my parents, Hayam and Ahmed, for their endless support every step of the way. They have carried me through every juncture of my life, and without them, I would not be writing this PhD thesis. I cannot thank enough my sister, Noha, for her endless support and pushing me forward, and my brother, Mohammad, for his continuous support and care.

Dedication

To my parents, Hayam and Ahmed.
To my sisters and brother, Doaa, Noha and Mohammad.
To my family.

Contents

Publications iv

Abstract v

Acknowledgments vii

List of Figures, Tables and Algorithms xiii

Abbreviations and Notations xvi

1 Introduction 1

1.1 Motivations 1

1.2 Aims, Approaches and Contributions 4

1.2.1 Aims 4

1.2.2 Approaches 4

1.2.3 Summary of Contributions 7

1.3 Thesis Structure 9

2 Background 10

2.1 Basic Mathematical Concepts 10

2.1.1 Probability 10

2.1.2 Information Theory 11

2.1.3 Maximum Likelihood Estimation (MLE) 13

2.2 Deep Learning 15

2.2.1 Deep Generative Models 16

2.2.2 Sequential Deep Generative Models 22

2.3 Reinforcement Learning (RL) 26

2.3.1 Finite Markov Decision Processes (MDPs) . . . 26
2.3.2 Bellman Equations . . . 29
2.3.3 Main Approaches . . . 30
2.3.4 Policy Gradients . . . 32
2.3.5 Reducing Policy Gradients Variance . . . 34
2.4 Adversarial Examples . . . 36
2.4.1 Crafting an Imperceptible Attack . . . 37
2.4.2 Types of Adversarial Attacks . . . 38

3 Adversarial Autoregressive Networks **39**
3.1 Introduction . . . 40
3.2 Background . . . 41
3.2.1 Generative Adversarial Networks (GAN) . . . 41
3.2.2 Mode Collapse . . . 42
3.3 Proposed Framework . . . 43
3.3.1 Model Definition . . . 44
3.3.2 Training . . . 46
3.3.3 Relaxed argmax using Gumbel-Softmax . . . 46
3.3.4 Overall Training and Inference . . . 47
3.4 Experiments . . . 47
3.5 Summary . . . 52

4 Goal-Oriented Controlled Generation **55**
4.1 Introduction . . . 56
4.2 Related work . . . 58
4.3 Background . . . 59
4.3.1 Problems of Interest . . . 60
4.4 Proposed Framework . . . 62
4.5 Experiments . . . 65
4.5.1 Text generation . . . 66
4.5.2 Air-Combat Trajectory Generation . . . 69
4.6 Summary . . . 75

5 Adversarial Attacks via Cross-Domain Interpretability **77**

5.1 Introduction 78

5.2 Background 79

5.2.1 Adversarial Examples in Natural Language 80

5.3 Related Work 82

5.4 Proposed Framework 83

5.5 Experiments 88

5.6 Summary 93

6 Learning Adversarial Attacks with Target Model Guidance **95**

6.1 Introduction 96

6.2 Method 97

6.2.1 Substitute training without access to the target test set 98

6.2.2 Substitute training with access to the target test set 98

6.3 Experiments and Discussion 99

6.4 Conclusion 103

7 Conclusions and Future Directions **104**

7.1 Contributions 104

7.2 Future Directions 107

Appendices **109**

Bibliography **113**

List of Figures

2.1 The relationship between mutual information, marginal entropy, conditional entropy and mutual entropy. . . . 13
2.2 A recurrent neural network, folded in time. . . . 18
2.3 The Long-Short Term Memory (LSTM) cell. . . . 20
2.4 The GAN Model. G is the generator, and D is the discriminator 21
2.5 Latent space representation learned by VAE. . . . 22
2.6 Agent-environment interaction in RL. . . . 26
2.7 An illustration of a MDP. At each step, the agent takes an action that changes its state in the environment and provides a reward. . . . 28
2.8 Relations between main RL methods 31

3.1 Mode collapse problem on a 2-D mixture of Gaussians data. . . . 42
3.2 Behavior of minimizing KL divergence (left) and reverse KL divergence (right). 43
3.3 Proposed Adversarial Autoregressive Network 44
3.4 Training inputs and outputs for the ARN recurrent generator. . . . 44

4.1 "Stern Conversion" Flight Maneuver. . . . 61
4.2 Overview of OptiGAN framework. . . . 62
4.3 NLL values on MS-COCO Dataset. Unlike SeqGAN, OptiGAN does not sacrifice output diversity. . . . 69
4.4 Samples of the training data and generated trajectories from the model. . . . 72
4.5 Samples of the training data and generated trajectories from the conditional model. . . . 73
4.6 Examples of novel behaviour generated by the model different from training data. . . . 74

5.1 The standard procedure to craft a natural language adversarial example . . . 80
5.2 Overview of Explain2Attack Framework. 84

6.1 Overview of Explain2Attack L2X training with Target Guidance via a) the substitute sentences, or b) the target test set. 99
6.2 Adversarial Accuracies for Explain2Attack with the selector trained on a portion of the target test set. 102
6.3 Average Queries for Explain2Attack with the selector trained on a portion of the target test set. 103

List of Tables

3.1 BLEU, FC and Diversity scores for IMDB Reviews dataset 50
3.2 BLEU, FC and Diversity scores for MS-COCO dataset 50
3.3 BLEU, FC and Diversity scores for the small EMNLP dataset 50
3.4 Generated sample sentences of ARN trained on IMDB dataset 51
3.5 Generated samples from SeqGAN on MS-COCO dataset. 51
3.6 Low/average BLEU samples from ARN on MS-COCO dataset 52

4.1 Comparison Baselines 65
4.2 BLEU scores and NLL values on MS-COCO dataset 67
4.3 BLEU scores and NLL values on EMNLP News 2017 Dataset 67
4.4 Generated sample sentences from our model 70
4.5 Sentences from OptiGAN-OnlyRL. Pure RL looses structure with BLEU rewards, repeating certain n-grams 70
4.6 Blue Fighter Engagement Scores (McGrew Score) 71
4.7 Effect of hyper-parameter λ for GAN-Only training 71

5.1 Statistic of Used Datasets 89
5.2 After-Attack Accuracies, Queries and Query Efficiency 90
5.3 Effect of Sentence Length on Number of Queries 91
5.4 Perturbation Query Cost **(PQC)** 92
5.5 Adversarial Examples Sentences. Perturbed Words are Highlighted 92

6.1 Performance metrics for Explain2Attack selector trained with target model predictions given substitute dataset sentences. 100
6.2 Performance metrics for the selector trained with target predictions given target test set sentences (5K out of 25K target test sentences used). 101

List of Algorithms

1 REINFORCE algorithm 33

2 ARN Training 47

3 ARN Inference 47

4 OptiGAN Monte Carlo rollout to reduce REINFORCE variance 65

5 Train Substitute Model SUB 86

6 Explain2Attack: Crafting Text Adversarial Examples 87

Abbreviations

Abbreviation	Description
p.d.f / p.m.f	probability density function / probability mass function
KL / JS	Kullback-Leibler / Jensen-Shannon
ML / MLE	maximum likelihood / maximum likelihood estimation
GAN	generative adversarial network
VAE	variational autoencoder
MDP	Markov decision processes
RL	reinforcement learning
MC	Monte Carlo

Notations

Notation	Description
X, x	If X is a random variable, then x is a realisation of X.
P, p	If a probability distribution is denoted by an upper-case letter (e.g. $P(\cdot)$), then its p.d.f or p.m.f is denoted by the corresponding lower-case letter (e.g. $p(\cdot)$). However, when ambiguity is not an issue based on the context, we will interchangeably use the upper-case letter for both distribution and its p.d.f/p.m.f.
$\mathbb{E}_p[X]$	The expectation of a random variable X under probability distribution P
$KL(\cdot)$, $JS(\cdot)$	Kullback-Leibler / Jensen-Shannon divergences
$\mathcal{N}(\mu, \sigma^2)$	A Gaussian (or normal) distribution with mean μ and variance σ^2
π	reinforcement learning "policy" probability distribution
S_t, A_t, R_t	In a Markov Decision Process (MDP): S_t is the state of environment, A_t is the action taken by the agent, R_t is the reward received by the agent from the environment, all at time step t.
G_t	The accumulated future rewards an agent receives at time step t.

Chapter 1

Introduction

1.1 Motivations

Teaching machines to process and understand sequence data is a critical component for advanced artificial intelligence systems. In many artificial intelligence applications, including natural language understanding, drug discovery, and autonomous vehicles, sequences of various data types are used to learn and execute certain tasks. For example, natural language and speech are sequences of words or utterances, in robot motion planning, a trajectory is an action sequence learned from experiences or sensory data. Drugs or materials are designed from chemical graph structures represented as sequences, and music is composed of a sequence of sound keynotes.

There has been extensive progress towards building machine learning models for processing sequence data. However, most of these models were mainly used for classification or prediction tasks. Until recently, sequence models lacked the ability to become creative, or to accurately learn the given data statistical distribution and generate new similar data. They also lacked a common sense understanding of the sequences, like natural language, where malicious input could easily confuse a highly accurate classification model. Hence, in order for artificial intelligence agents to be able to carry out advanced tasks, building such lacking capabilities is crucial, and will have a significant impact to many real-world applications.

In the last decade, deep learning has become the dominant approach for efficient learning from large amounts of data. Along with this success, more sophisticated deep models have been developed for various tasks. Nonetheless, employing deep learning to build AI systems

for previously described applications faces two difficulties; **density estimation**, and **deep models robustness,** which are the focus of this thesis.

The problem of density estimation is concerned with learning the underlying data distribution. Once a data distribution is estimated by an appropriate model, it can then be used to generate new samples that resemble the original data distribution. **Deep generative models** like **Variational Autoencoders (VAE)** (Kingma and Welling, 2013) or **Generative Adversarial Networks (GAN)** (Goodfellow et al., 2014a) are deep density estimators, that can be used to generate new data in advanced applications like drug discovery and natural language generation. Although deep generative models were successfully used for various computer vision tasks, its application for sequence generation remains a challenge, specially for discrete data like natural language.

Moreover, despite the success of deep learning in terms of accuracy metrics, research has shown that common deep learning models can be easily fooled by malicious input that looks like the training data (Goodfellow et al., 2014b), but slightly perturbed, in a way imperceptible to humans. These perturbed inputs are called **adversarial examples**, which can be used to attack trained models, causing significant deterioration to the model's performance. Needless to say, deploying such models in real-world applications without enough understanding of its behaviour is an issue of critical concern. For instance, a computer vision system of a self-driving car can be easily fooled by a slight corruption in the input frame, or an unusual visual pattern of a human dress or an object. Such vulnerability can easily lead to confused decisions by the car's system, and eventually to fatal accidents. Interestingly, researchers have found that the better we understand how a model is vulnerable to different attacks, the better we can increase its robustness. For instance, augmenting generated adversarial examples in the training data can improve robustness of models (Goodfellow et al., 2014b). Therefore, evaluating and improving the robustness of deep learning models against adversarial inputs is essential for many real-world applications.

My thesis aims to address these deep learning difficulties and shortcomings for sequence data, and to advance deep learning models towards aforementioned real-world applications. Specifically, my thesis addresses three topics; generative models *diversity*, *controlled generation*, and generating *adversarial attacks* on sequence models.

- **Generative Models Diversity**. Deep generative models like Generative Adversarial

Networks (GAN) suffer from a well-known problem of **mode-collapse** (Goodfellow, 2016), which causes GANs be incapable of generating diverse outputs when learning from data with underlying multi-modal distribution. When mode-collapse happens during training, GANs fail to learn many different modes of underlying data distribution, and only learns few modes. For real-world applications like natural language generation or drug discovery, diverse outputs are desired to help convey human-like conversation, or explore different useful drug leads for lab experiments. Existing GAN sequence generative models like SeqGAN and MaskGAN (Yu et al., 2017; Fedus et al., 2018) suffer from the same mode-collpase problem, and do not offer a principled approach to overcome it. This part of the thesis addresses the diversity problem in sequence GANs.

- **Controlled Generation for Sequence Models**. Existing generative models mainly generate sequences to closely mimic the training data, without much control on what the desired output from the model should be. Therefore, these models have been mainly used for artistic or entertainment applications. For certain real-world applications, such as natural language generation (Hu et al., 2017), material or drug design (De Cao and Kipf, 2018; Guimaraes et al., 2017; Putin et al., 2018; Polykovskiy et al., 2018), or autonomous motion planning, we are not only interested in generating data similar to the real ones, but we need them to have specific useful properties or attributes. The main challenge that hinders the application of generative models in such domains is the absence of mechanisms to optimize the generated outputs according to certain metrics or useful properties. In this part of the thesis, we address the controlled generation issue of sequence GANs, where we seek to introduce a useful mechanism to optimize the generative model towards generating outputs with desired properties.

- **Adversarial Attacks on Sequence Models**. As previously discussed, training robust deep learning models for downstream tasks is a critical challenge. Equally challenging is finding those adversarial examples that most severely damage the model's performance. In a typical adversarial attack setting, the model attacked with malicious input is called the *target model*, while the algorithm that generates the adversarial examples is called the *attacker*. In general, attacks using adversarial examples can be crafted in either white-box or black-box settings. In ***white-box*** attacks, the attacker

has access to the target model parameters, and the gradient of these parameters is used to craft adversarial examples (Belinkov and Glass, 2018; Wang et al., 2019; Yang et al., 2019; Sato et al., 2018). On the other hand, ***black-box*** attacks do not have access to the model parameters (Kuleshov et al., 2018; Gao et al., 2018; Jin et al., 2019), but only to its outputs. In this part of the thesis, we address the problem of generating natural language processing (NLP) adversarial attacks, and we focus on the black-box setting, since in practice, this is the more probable scenario for AI systems deployed in the real-world.

1.2 Aims, Approaches and Contributions

1.2.1 Aims

The fundamental aims of this dissertation are to advance the knowledge in sequence generative modeling and adversarial attacks on sequence models. In particular, there are two main objectives:

- *Developing new models to address diversity and controlled generation problems in unsupervised sequence generation.*
- *Developing new algorithms to tackle the problem of generating adversarial attacks on natural language models.*

1.2.2 Approaches

The different approaches we propose towards achieving the presented objectives above fall into three topics. Below we describe briefly these topics and the proposed approaches.

1.2.2.1 Adversarial Autoregressive Network (ARN) for Diverse Sequence Generation

Existing GANs for sequence generation (Yu et al., 2017; Fedus et al., 2018) suffer from the mode-collapse issue. Because of this problem, the GAN generator model tends to generate samples that lack diversity, learned from few modes of training data, ignoring other modes that were missed during training. There are different reasons that contribute to this problem

(Goodfellow, 2016), including; i) the problem of catastrophic forgetting, a well-known problem of deep neural networks, where the network tends to forget previously learned tasks, ii) the GAN training procedure, and iii) the nature of GAN objective function used in practice. In this part of the thesis, we address the problem of diversity and mode-collapse from the angle of GAN objective function. In details, we change the standard GAN objective to maximize the log-likelihood of data and minimize the Jensen-Shanon divergence between data and model distributions, simultaneously. This way, we encourage a balance between the "mode-seeking" behaviour of GAN and the "mode-covering" behavior of maximum likelihood estimation.

Furthermore, in order to incorporate compressed latent representation in the generative model, we introduce a latent variable at the first token, and maximize its variational lower bound. Incorporating a latent space in the generative model allows learning high level representations from the input into a smaller more abstract latent space. This can be later helpful in controlling the generation process through changing/fixing specific factors of the latent space according to the desired outcome . While our work is demonstrated with discrete data, it can be straightforwardly adopted for continuous data.

1.2.2.2 Goal-Oriented Sequence Models for Controlled Generation

To realize the full potential of generative models in real-world applications, generative models need to have mechanisms to optimize the generated outputs according to certain metrics or useful properties. In this part of the thesis, we aim to incorporate a controlling mechanism that allow controlled generation for sequence GANs. We propose to incorporate an optimisation mechanism that can be given explicit scores or metrics that need to be optimized by the generative model. This capability can be then used in applications like natural language generation or autonomous vehicles to allow the generative model to achieve the best possible scores for natural language realism or motion trajectory safety or maneuvering advantage. One of the well-established frameworks to achieve for this purpose is **Reinforcement Learning (RL)**. RL is sequential decision optimisation framework, that can learn suitable policies to maximize desired domain rewards. However, through various experiments, we show that applying pure RL techniques or RL-based GANs (Yu et al., 2017; Fedus et al., 2018) for sequence generation suffers from severe mode-collapse problem, that

makes the optimisation framework moves away from the true data distribution. To address this key problem, we extend our model from first part (ARN) by incorporating an additional RL optimisation objective. This reward is intended to be chosen by the user to encourage model optimisation towards certain properties or attributes, according to the desired task of the model. By combining both GAN and RL in a simple and general framework, we achieve both the addition of a controlling mechanism to the generative model, while preserving the training of the sequence GAN to stay close to the true data distribution. We name our proposed framework *OptiGAN*.

We demonstrate the performance of OptiGAN through comprehensive experiments in two applications: text generation (discrete data) and air combat trajectory generation (real-valued data). For text generation task, we aim to generate sentences resembling real sentences in a given text corpus, while optimizing the BLEU (Papineni et al., 2002) score for obtaining better quality natural language sentences. For aircraft trajectory generation task, we aim to generate a trajectory plan for air-combat maneuver scenario between two aircraft and a certain engagement score, which reflects the tactical quality of aircraft trajectories in an air combat.

1.2.2.3 Black-Box Adversarial Attacks on Sequence Models

In this part of the thesis, we address the problem of generating attack on natural language classification task in the black-box setting. Typical classification models such as deep neural networks (DNN) output a probability distribution of their input belonging to each target class. Usually, the final label of the model is decided to be the one with the maximum probability. Hence, a classification model could be fooled if the confidence of the output probability is affected by a malicious input, switching the maximum probability to another incorrect target class. The key strategy used to generate adversarial text in existing methods is to try to replace few important words in an input sentence with synonyms such that its meaning remains the same. The classification model is then queried with these perturbed sentences to find out which ones successfully change the output label. Existing state-of-the-art models have different ways to search for most important words to replace (Jin et al., 2019; Ren et al., 2019; Gao et al., 2018), but the common intuition is to compute the importance score for each word as a function of the probability output of target model. This means

that, for each word in a sentence, they query the target model to retrieve an importance score for this particular word. This word by word querying of the target model is costly in both computational complexity and number of queries. For real world scenarios, the number of queries is critical, where less queries are desired to avoid suspicion towards an attacking agent.

We propose a more efficient approach to the word importance ranking problem. Instead of searching for important words to be perturbed by querying the target model, we use *interpretable models across domains* to *learn* word importance scores. The key idea is to replace the need to querying the target model by learning a similar substitute model with similar domain data. Then, this trained substitute model can be used to generate word importance scores for the targeted model. Therefore, our novel framework, named *Explain2Attack*, potentially eliminates a significant query and running time complexity from the attacking procedure. We conduct extensive experiments on attacking state-of-the-art models using novel framework, where we designed set of experiments to evaluate the query efficiency and quality of the framework compared to the competitor baseline.

1.2.3 Summary of Contributions

This thesis contributes novel learning methods through extensive experiments to key issues in deep sequence modeling:

- We propose a new generative framework for sequence data called *Adversarial Autoregressive Networks (ARN)* to address the problem of low output diversity in sequence GANs. By changing the training objective of standard GANs in a principled way, we prove that ARN approximates the true underlying data distribution. In natural language generation task, we conducted experiments to compare ARN to existing GAN sequence generation baseline, and we empirically show that ARN achieves higher diversity scores compared to the baseline.

- Unlike existing unsupervised sequence generation GANs for discrete data, ARN incorporates a latent space through variational lower bound on the first token. This capability helps make ARN useful for applications where high level representations from the data can be learned. These learned representations can then be used for controlling the generated outputs through changing/fixing specific factors of the latent

space according to the desired outcome (e.g. generating natural language sentences with a certain sentiment or tense, or generating chemical molecules that contain desired properties).

- We propose a novel deep generative model, OptiGAN, that employs reinforcement learning as a control mechanism to maximize desired domain rewards. OptiGAN is specifically useful for controlled generation applications, where the model is required to maximize certain score or metric for its generated output (e.g. generating a chemical molecule with high solubility, activity and ease of synthesis). OptiGAN benefits from the diversity improvement used in our first model, Adversarial Auto-regressive Networks (ARN), where it is trained with the same improved objective function. Via comprehensive experiments in two different tasks, we evaluate both the diversity and optimized scores of generated outputs from OptiGAN against other sequence generative models. We found that OptiGAN achieves higher scores than baseline models aided by the RL component, while preserving the diversity of generated outputs.

- To the best of our knowledge, OptiGAN is the first GAN controlled generative model for sequences that addresses the diversity issue in a principled approach. OptiGAN combines the benefits of GAN and RL policy learning, while avoiding their mode-collapse and high variance drawbacks.

- With comprehensive studies, we empirically show that if only pure RL is applied to maximize a score of interest, that the realism of the output might be sacrificed for the sake of superficially obtaining higher scores. For instance, in the case of text generation, the model was able to cheat the selected quality score by generating sentences in which few words are repeated all the time. This shows that combining a GAN-based objective with RL encourages the optimisation procedure of RL to stay close to the true data distribution.

- We propose a novel adversarial attacks framework, *Explain2Attack*, an efficient framework to generate black-box adversarial attacks on natural language models. Explain2Attack is able to achieve or out-perform state-of-the-art methods in attack rates, yet with reduced number of queries, where the number of queries is of critical concern for an attacking agent. Moreover, our framework scales significantly better in terms

of queries with the length of input sequences. To the best of our knowledge, Explain2Attack is the first framework that learns word importance scores, instead of searching for them using traditional expensive procedures. Our framework advances the state-of-the-art in this area, where computationally expensive word ranking procedures are not needed any more to generate successful adversarial attacks.

1.3 Thesis Structure

The remaining part of this thesis is structured as follows:

- Chapter 2: *Background.* We briefly present the essential background that lays the foundations for all our work in this thesis.
- Chapter 3: *Adversarial Autoregressive Networks.* In this chapter, we discuss our work on the first topic to address diversity issue in generative sequence models.
- Chapter 4: *Goal-Oriented Controlled Generation.* We discuss our work on controlled generation for sequence models using reinforcement learning.
- Chapter 5: *Adversarial Attacks via Cross-Domain Interpretability.* We present our work on learning substitute models and generating adversarial attacks on natural language classification models in a black-box setting.
- Chapter 6: *Learning Adversarial Attacks with Target Model Guidance.* We extend our work in Chapter 5 to employ target model outputs to improve substitute learning and adversarial attacks on natural language models.
- Chapter 7: *Conclusions and Future Work.* Finally, we summarize all scientific contributions of this thesis as well as potential future directions.

Chapter 2

Background

In this chapter we provide fundamental concepts behind our work. First, we describe basic probability and information theory concepts commonly used in machine learning. Second, we provide the basic background on deep learning, generative and sequence models relevant to the thesis. Third, we provide an overview of reinforcement learning fundamental concepts and approaches relevant to work in Chapter 4. Finally, we provide a formal introduction to adversarial examples relevant to Chapter 5.

2.1 Basic Mathematical Concepts

2.1.1 Probability

A divergence in statistics between two probability distributions P and Q quantifies the information difference between them. We define below the common divergences in machine learning that are used throughout the thesis.

Definition 1 (Kullback-Leibler divergence)**.** Given a random variable X and two probability density functions (or two probability mass functions) $p(X)$ and $q(X)$, the **Kullback-Leibler divergence** of the distribution Q from the reference distribution P is defined as:

$$KL(P \parallel Q) \triangleq \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} = \mathbb{E}_p \left[\log \frac{p(x)}{q(x)} \right].$$

From the definition, we can see that KL divergence is asymmetric.

Definition 2 (Reverse KL divergence)**.** For two probability distributions P and Q, the **Reverse KL divergence** is defined as:

$$RKL(P \parallel Q) \triangleq \sum_{x \in X} q(x) \log \frac{q(x)}{p(x)}.$$

It is easy to see that $KL(P \parallel Q) \neq RKL(P \parallel Q)$. In many machine learning algorithms, either KL or RKL are used as the optimisation objective, where one distribution is the empirical data distribution, and the other is the model distribution that is desired to be learned .

Definition 3 (Jensen-Shannon divergence)**.** The Jensen Shannon divergence between two distributions P and Q is defined as:

$$JS(P \parallel Q) \triangleq \frac{1}{2} KL\left(P \parallel \frac{1}{2}(P+Q)\right) + \frac{1}{2} KL\left(Q \parallel \frac{1}{2}(P+Q)\right).$$

Lemma 4 (Jensen's inequality)**.** *If p is a p.d.f and f is a convex function, then we have $\mathbb{E}_p[f(x)] \geq f(\mathbb{E}_p[x])$.*

If f is concave, we have $\mathbb{E}_p[f(x)] \leq f(\mathbb{E}_p[x])$.

The equality holds if and only if f is not strictly convex or X is constant.

A proof of Lemma 4 can be found in the solutions to Problems 6.1 (when p is a p.m.f) and 7.5 (when p is a p.d.f) in (Steele, 2004).

2.1.2 Information Theory

Definition 5 (Shannon Entropy)**.** For a discrete random variable X whose p.m.f is $p(X)$, the **Shannon entropy** is defined as:

$$H(X) \triangleq -\mathbb{E}_p[\log p(x)] = -\sum_x p(x) \log p(x).$$

The **differential entropy** (loosely considered as the continuous counterpart of Shannon entropy) of continuous random variable X following p.d.f $f(X)$ is defined as:

$$H(X) \triangleq -\mathbb{E}_f[\log f(x)] = -\int f(x) \log f(x) \, \mathrm{d}x.$$

Definition 6 (The Joint Entropy)**.** For two discrete random variables X, Y, the joint entropy is:

$$H(X,Y) \triangleq -\sum_x \sum_y p(x,y) \log p(x,y).$$

Entropy is additive for independent random variables:

$$H(X,Y) = H(X) + H(Y) \quad \text{iff } p(x,y) = p(x)p(y).$$

The conditional entropy of X given Y defined as:

$$\begin{aligned} H(X \mid Y) &\triangleq -\sum_y p(y) \left[\sum_x p(x \mid y) \log \frac{p(x,y)}{p(y)} \right] \\ &= -\sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(y)}, \end{aligned}$$

which measures the average uncertainty that remains about x when y is known.

Lemma 7 (Chain rule for entropy)**.** *The joint entropy, conditional entropy and marginal entropy are related by:*

$$H(X,Y) = H(X) + H(Y \mid X) = H(Y) + H(X \mid Y),$$

which means that the uncertainty of X and Y is the uncertainty of X plus the uncertainty of Y given X.

Definition 8 (Mutual Information)**.** The mutual information between X and Y is defined as:

$$I(X;Y) \triangleq H(X) - H(X \mid Y),$$

and satisfies both equality $I(X;Y) = I(Y;X)$, and non-negativity $I(X;Y) \geq 0$ properties. It measures the average reduction in uncertainty about x that results from learning the value of y; or vice versa, the average amount of information that x conveys about y.

The mutual information between X and Y, is also defined as the KL divergence of the product of their marginal distributions from their actual joint distribution:

$$I(X;Y) \triangleq KL\Big(p(x,y) \parallel p(x) \cdot p(y)\Big).$$

Similarly, the mutual information among n random variables $X_1, X_2, \ldots X_n$ is

$$I(X_1; X_2; \ldots; X_n) \triangleq KL\left(p\left(x_1, x_2, \ldots x_n\right) \parallel \prod_{i=1}^{n} p\left(x_i\right)\right),$$

which is the divergence of the product distribution from the joint distribution.

It is also easy to show that:

$$\begin{aligned} I(X;Y) &= H(X) + H(Y) - H(X,Y) \\ &= H(X) - H(X \mid Y) \\ &= H(Y) - H(Y \mid X). \end{aligned}$$

Figure 2.1 shows how the total entropy $H(X,Y)$ of a joint ensemble can be broken down.

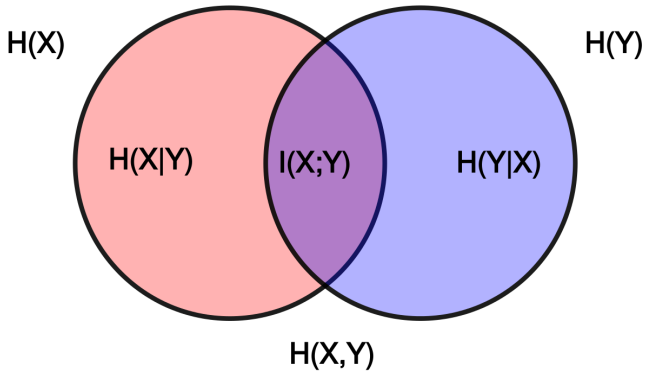

Figure 2.1: The relationship between mutual information, marginal entropy, conditional entropy and mutual entropy.

2.1.3 Maximum Likelihood Estimation (MLE)

Maximum likelihood is a fundamental principle in machine learning that allows to learn a probability distribution that approximately estimates the empirical distribution of some training data. Formally, given a dataset $D = \{x_1, x_2, \ldots, x_N\}$ consisting of a set of N data points x, we can define a model with probability distribution P_{model}, parameterized by some parameters θ, that approximately estimates the empirical data probability distribution P_{data}. We then refer to the likelihood as the probability that the model assigns to training the dataset D. Maximum likelihood simply means to find the optimal parameters θ^* for

P_{model} that maximize its likelihood on the training data :

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} p_{\text{model}}(x_i; \theta).$$

Because of the numerical problems of decimal operations in digital computers, such as underflow resulting from multiplying together several very small probabilities, this computation is easier to do in the log space. The likelihood computed in the log space is called the ***log-likelihood***, where a sum is computed rather than a product over probabilities, which is less prone to numerical problems:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{N} \log p_{\text{model}}(x_i; \theta).$$

Since the maximisation procedure does not change when the cost function is multiplied by a constant, the inner optimisation function can be divided by the number of points N, and it then becomes an expectation with respect to the empirical data distribution P_{data} (under the assumption that the data points are independent and identically distributed (i.i.d.)):

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} \log p_{\text{model}}(x_i; \theta) = \underset{\theta}{\operatorname{argmax}} \underset{x \sim p_{\text{data}}}{\mathbb{E}} [\log p_{\text{model}}(x; \theta)]. \tag{2.1}$$

Lemma 9. *Maximizing Maximum Likelihood corresponds to Minimizing KL Divergence between P_{data} and P_{model}*

Proof. Minimizing the KL divergence between P_{data} and P_{model} is expressed as :

$$\begin{aligned}
\theta^*_{KL} &= \underset{\theta}{\operatorname{argmin}}\ KL(P_{\text{data}} \parallel P_{\text{model}}) \\
&= \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{x \sim P_{\text{data}}} \left[\log \frac{p_{\text{data}}(x)}{p_{\text{model}}(x; \theta)} \right] \\
&= \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{x \sim P_{\text{data}}} [\log p_{\text{data}}(x) - \log p_{\text{model}}(x; \theta)] \\
&= \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{x \sim P_{\text{data}}} [\log p_{\text{data}}(x)] - \mathbb{E}_{x \sim P_{\text{data}}} [\log p_{\text{model}}(x; \theta)] \\
&= \underset{\theta}{\operatorname{argmin}}\ H(P_{\text{data}}) - \mathbb{E}_{x \sim P_{\text{data}}} [\log p_{\text{model}}(x; \theta)],
\end{aligned}$$

where that the last line contains the empirical distribution entropy $H(P_{\text{data}})$, which does not depend on the parameters θ being optimized. Hence, the entropy term can be discarded and the optimal parameters under KL minimisation becomes:

$$\begin{aligned}\theta^*_{KL} &= \underset{\theta}{\text{argmin}} \; -\mathbb{E}_{x\sim P_{\text{data}}}\left[\log p_{\text{model}}\left(x;\theta\right)\right] \\ &= \underset{\theta}{\text{argmax}}\; \mathbb{E}_{x\sim P_{\text{data}}}\left[\log p_{\text{model}}\left(x;\theta\right)\right],\end{aligned}$$

which is the same as the maximum likelihood estimation in Eq.(2.1). □

2.2 Deep Learning

In the last decade, deep learning has become the dominant approach for effectively learning from data in real-world applications on large scale. This is due to the nature of neural networks as ***universal function approximators*** (Hornik et al., 1989), which means that given two independent variables x and y related with some arbitrary unknown function $f(x)$, there always exists a neural network f_θ is parameterized by some parameters θ that can closely approximate f. The closeness of this approximation depends on the probability distribution of the data and the architecture and depth of the neural network.

Recently there has been increased research activity to discover the most suitable network architectures for different data types. For example, Convolutional Neural Network (CNN) (Fukushima and Miyake, 1982; LeCun et al., 2015) is the standard state-of-the-art architecture for computer vision tasks, while auto-regressive and recurrent architectures like Recurrent Neural Networks (RNN) (Rumelhart et al., 1986; LeCun et al., 2015; Hochreiter and Schmidhuber, 1997) are most suitable for sequential and time series tasks such as Natural Language Processing (NLP). Attention mechanisms used in the recent Transformer (Vaswani et al., 2017) architectures are increasingly successful in both sequential and vision tasks. In this section, we briefly describe the main deep neural network architectures and models commonly used for generative and sequential tasks.

2.2.1 Deep Generative Models

Deep generative models learn both observed and target variables, by estimating a joint probability distribution over these variables. For input data x and associated set of labels y, a generative model learns the joint probability distribution $p(x, y)$ which allows the model to generate new data that follow the same distribution of training data. Generally, in unsupervised deep generative models, input data x serves as both the observed and target variables for the model.

There are mainly two different types of deep generative models, where they differ in the way the estimate data probability density:

- **Explicit tractable density models.** These models try to explicitly maximize the log-likelihood of the data probability density function (Uria et al., 2016; Germain et al., 2015) . Deep neural autoregressive models like PixelRNN, PixelCNN, and WaveNet (Van Oord et al., 2016; Oord et al., 2016b;a) are recent successful examples of this type of models for sequential data. These models are flexible and powerful in the sense that they can model conditional relations on the level of components of individual data points, where each component can be conditionally defined given the previous components. However, these models do not incorporate latent variable learning, which allows learning higher representations of the data, that can be later used to control the generation process to produce specific desired features.

- **Approximate and implicit density models.** These models like Variational Autoencoders (VAE) (Kingma and Welling, 2014) and Generative Adversarial Networks (GAN) (Goodfellow et al., 2014a) try to approximate to the log-likelihood of the density function or indirectly estimate the probability distribution instead of explicitly targeting to estimate it. Unlike explicit density models, GANs and VAEs employ latent variables or noise space that can capture underlying meaningful representations of the data. GANs are particularly very efficient for high dimensional data like images (Zhang et al., 2019; Brock et al., 2019; Hoang et al., 2018; Radford et al., 2015), and provide very good sample quality compared to other generative models.

In this thesis, we mainly focus on the second family of generative models, like GANs and VAEs, where we can leverage the latent representation space for the sake of introducing a

richer and controllable mechanism for sequence generation.

2.2.1.1 Autoregressive and Recurrent Sequence Models

Neural autoregressive models are generative neural networks in which training and sampling are done in an autoregressive process. Autoregression means that during training or sampling, the current output is used as an input for the network to train or compute the next output, and this process continues starting from the first sample till the last sample in the sequence. Recently, neural autoregressive models received increased focus and several new models were introduced (Germain et al., 2015; Van Oord et al., 2016) that led to state-of-the-art performance in many tasks including image and speech generation (Van Oord et al., 2016; Oord et al., 2016b;a).

The basic idea of autoregressive modeling is the conditional modeling of data. The model predicts the current output conditioned on past outputs. The joint probability of an input sequence X of length T can be described using the chain rule as:

$$\begin{aligned} p\left(X\right) &= p\left(x_1,\, \ldots,\, x_T\right) \\ &= p\left(x_1\right)\, p\left(x_2|x_1\right)\, \ldots p\left(x_T|x_{1:T-1}\right) \\ &= p\left(x_1\right) \prod_{t=2}^{T} p\left(x_t|x_{1:\,t-1}\right)\,, \end{aligned}$$

where x_t is the token at time t of the sequence X. In this setting, the data likelihood is described by the product of conditionals $p(x_t \,|\, x_1, ..., \, x_{t-1})$ of all input data. This way each data point follows a conditional distribution over previous points in the chosen order.

Because of the complexity of modeling all these conditional probabilities for every point, some autoregressive models might use an additional learning signal called the "hidden-state" (h) . The hidden state is used to approximate lossy history of previous points, thus the likelihood under an autoregressive model becomes:

$$\begin{aligned} p(X) &= p\left(x_1\right)\, p\left(x_2|h_1\right)\, \ldots p\left(x_T|h_{t-1}\right) \\ h_t &= f\left(h_{t-1}, x_t\right), \end{aligned}$$

where f is a differentiable function, and h_0 is an initial state. This allows the network to model the sequential dependencies between data tokens. The training procedure is done sequentially by showing the network the inputs in order starting from the first point to the last. The network then captures the conditional distributions of the inputs inside its internal hidden layers structure. The *Recurrent Neural Networks* (RNNs) and *Long-Short Term Memory* (LSTMs) (Rumelhart et al., 1986; LeCun et al., 2015; Hochreiter and Schmidhuber, 1997) are standard examples of autoregressive models with recurrent connections. In this thesis, we mainly use recurrent models for developing the deep sequence models as they are simple to understand, well-established and easy to use compared to recent non-recurrent models. However, as we will discuss later in Chapters 3 and 4, our developed frameworks are general and can be adapted for use with other non-recurrent architectures.

Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) (Rumelhart et al., 1986; LeCun et al., 2015) are feedforward neural networks with the addition of hidden layer connections that connect adjacent time steps, introducing a notion of time or sequence to the model. Figure 2.2 shows this architecture folded in time, where at time t, nodes with recurrent connections receive input from the current data point x^t and also from hidden node values h^{t-1} in the network's previous state.

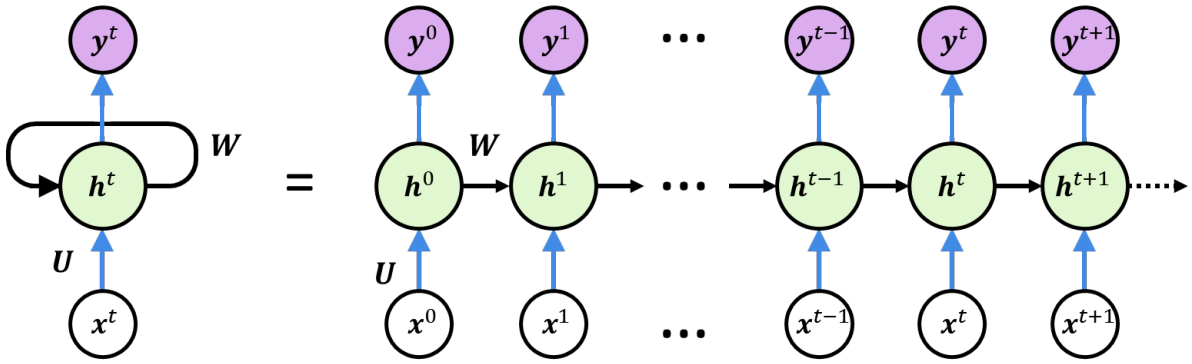

Figure 2.2: A recurrent neural network, folded in time. $U$and W are the weight matrices for input to hidden and recurrent hidden to hidden respectively

The output y^t at each time t is calculated given the hidden node values h^t at time t. Input x_{t-1} at time $t-1$ can influence the output at time t and later by through the recurrent connections. The forward pass for the hidden layer in a simple recurrent neural network is:

$$h_t = \sigma\left(Wh_{t-1} + Ux_t\right)$$
$$y_t = \tanh\left(Vh_t\right),$$

where U is the matrix of weights between the input and the hidden layer, W is the matrix of recurrent weights between the hidden layer and itself at adjacent time steps, and V is the matrix of weights between hidden layer and output. The network can be interpreted as a deep network with one layer per time step and shared weights across time steps. It is then clear that the unfolded network can be trained across many time steps using backpropagation through time (BPTT) (Werbos, 1990).

Learning with recurrent networks has long been considered to be difficult because of the challenge of learning long-range dependencies, as described in (Bengio et al., 1994; Hochreiter et al.). The problems of vanishing and exploding gradients occur when backpropagating errors across many time steps, where the gradient value either diminishes to a very small value or explodes to a very large value during training.

Long-Short Term Memory (LSTM)

Long Short-Term Memory, or LSTM (Hochreiter and Schmidhuber, 1997), was introduced in order to overcome the problem of vanishing gradients. This model resembles a standard recurrent neural network with a hidden layer, but each ordinary node in the hidden layer is replaced by a memory cell as shown in Figure 2.3. Each memory cell contains a node with a self-connected recurrent edge of fixed weight one, ensuring that the gradient can pass across many time steps without vanishing or exploding.

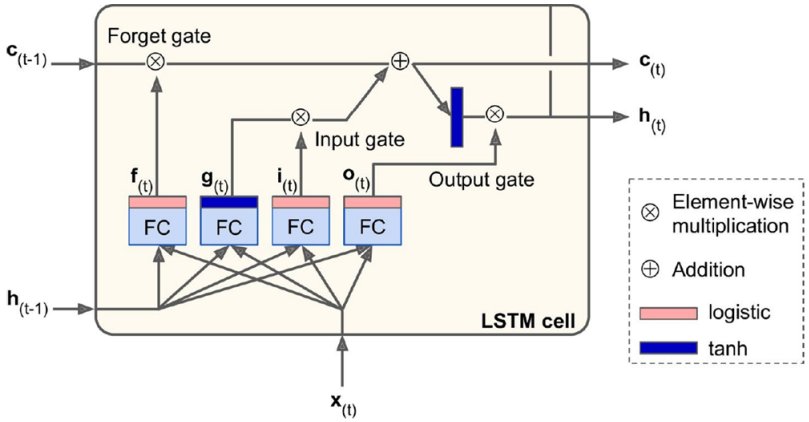

Figure 2.3: The Long-Short Term Memory (LSTM) cell. (Géron, 2017)

The term "long short-term memory" comes from the following intuition. Simple recurrent neural networks have long-term memory in the form of weights. The weights change slowly during training, encoding general knowledge about the data. They also have short-term memory in the form of temporary activations, which pass from each node to successive nodes. The LSTM model introduces an intermediate type of storage via the memory cell. A memory cell is a composite unit, built from simpler nodes in a specific connectivity pattern, with the novel inclusion of multiplicative nodes, represented in diagrams by symbol $\otimes$. The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state can be seen as a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions, which makes it's very easy for information to just flow along it unchanged. This gives LSTM the ability to remove or add information to the cell state, carefully regulated by structures called gates. LSTM networks have been shown to learn long-term dependencies more easily than the simple recurrent architectures on sequence processing tasks with state-of-the-art performance (Graves, 2012; 2013; Sutskever et al., 2014).

2.2.1.2 Latent Space Models

Generative Adversarial Networks (GAN) The basic idea of Generative Adversarial Networks (GAN) (Goodfellow et al., 2014a) is the adversarial training between two players, as shown in Figure 2.4. The goal of the first player, the generator G , is to get very good at

generating data that is very close to the real data that comes from real distribution $p_d(x)$. The goal of the second player, the discriminator D, is to distinguish real data from fake data generated by the generator. The standard GAN objective to optimize is the minimax game between D and G:

$$\min_G \max_D \left(\mathop{\mathbb{E}}_{x \sim p_d} \left[\log D\left(x\right) \right] + \mathop{\mathbb{E}}_{z \sim p_z} \left[\log \left(1 - D\left(G\left(z\right)\right)\right)\right] \right), \tag{2.2}$$

where z is the random noise input to G, and p_z is the prior distribution of the z. After the training is finished, the generator is used to generate data from any random input z.

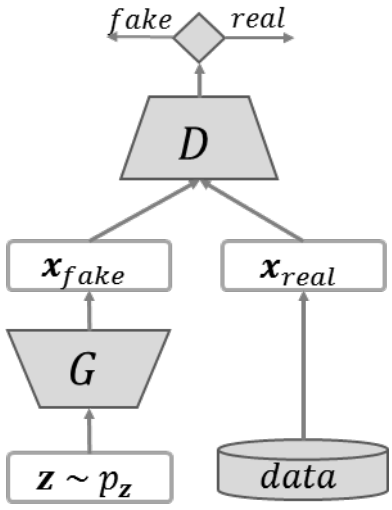

Figure 2.4: The GAN Model. G is the generator, and D is the discriminator

Variational Autoencoders (VAE) Variational Autoencoders (VAEs) (Kingma and Welling, 2014) approximate the maximum log-likelihood and can be trained using gradient descent. VAEs are trained to maximize a variational lower bound L on log-likelihood:

$$L(x\,;\theta) = \mathop{\mathbb{E}}_{z \sim q(z|x)} \left[\log p_{model}(x|z) \right] - KL\left(q(z|x) \parallel p_{model}(z) \right),$$

where $q(z|x)$ is a posterior and $p_{model}(z)$ is a prior distribution for the latent variable z. The first term is the data reconstruction likelihood. The second term works a regularizer to make

$q(z|x)$ and $p_{model}(z)$ close to each other. $p_{model}(z)$ can be chosen as $\mathcal{N}(0, I)$. $p_{model}(x|z)$ is the decoder, modeled as a neural network that resembles reconstruction of x from z sampled from the learned $q(z|x)$.

The latent variable The variable z is called a "*latent*" variable or code because it can capture implicit high level information or representations from the data. Such representations might be the face rotation on one dimension of the latent variable, while another dimension might capture the emotional expression as shown in Figure 2.5. Such latent variable space is useful for controlling the output of the decoder towards specific attributes like certain rotation and emotion by navigating or interpolating through the multi-dimensional space of z.

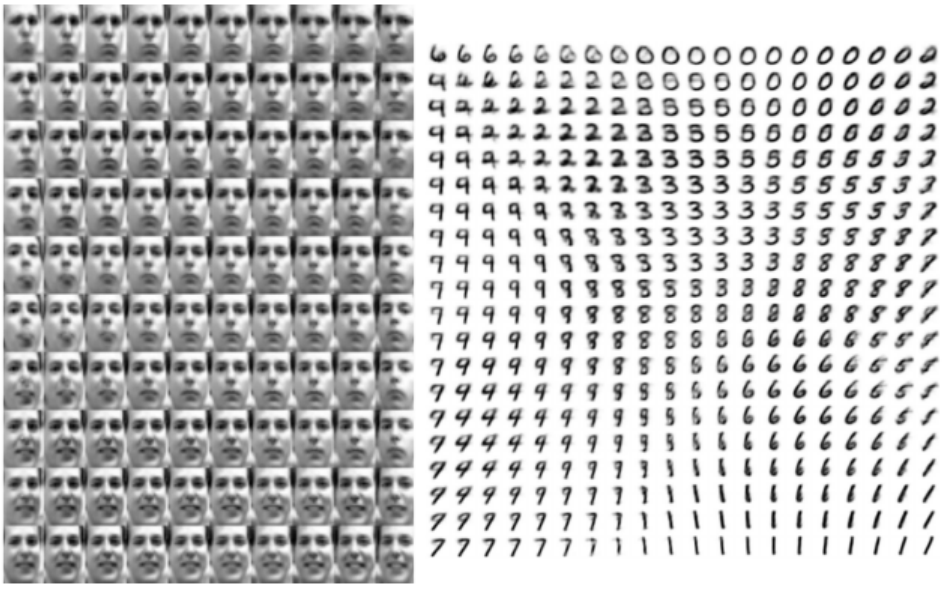

Figure 2.5: Latent space representation learned by VAE. Each image corresponds to a different choice of code z on a 2-D uniform grid. *(Left)* The 2-D map of the faces. One dimension that has been discovered (horizontal) mostly corresponds to a rotation of the face, while the other (vertical) corresponds to the emotional expression. *(Right)* The 2-D learned latent map of MNIST. (Goodfellow et al., 2016)

2.2.2 Sequential Deep Generative Models

Deep generative models including variational autoencoders and generative adversarial networks where successfully used for many computer vision tasks and with visual data like

images (Radford et al., 2015; Gulrajani et al., 2017; Zhang et al., 2019; Razavi et al., 2019). However, less attention was directed towards using VAEs and GANs for sequential data. To begin with, the architecture for these networks depended mainly on feed-forward layers like convolutional layers, which are originally designed for computer vision tasks, and do not necessarily perform well on sequence data. Sequence data like natural language or time series data have an additional inter-dependency between data points across time steps. Therefore, an appropriate type of layers like recurrent or attentional layers can be a more suitable building block for sequential generative models. In addition, for the particular case of discrete sequence data like natural language, applying standard VAEs and GANs faced major difficulties regarding gradient computation through the network parts and hyperparameters tuning. These shortcomings motivated research into developing sequential variants of VAEs and GANs that can be reliably used with sequence data including natural language or chemical and biological sequences.

With the increased attention for developing better sequential generative models, different approaches have been developed to achieve this goal. In general, the current sequential generative models are either based on the variational approximation principle of VAE or on adversarial training framework of GAN. Models based on variational approximation (Chung et al., 2015; Le et al., 2018a; Fraccaro et al., 2016; Roberts et al., 2018) are mainly based on recurrent autoregressive models like Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), incorporated into VAE training framework. These models were applied to many sequence generating tasks including handwriting and music generation. However, training VAE based models with autoregressive networks suffers from the problem of "*posterior collapse*" (He et al., 2019), where the latent variables are often ignored, especially when trained for discrete data like text (Bowman et al., 2016). In addition, the utilisation of learned latent space for controlled reconstruction is relatively less studied for sequential data.

Sequential GAN-based models, on the other hand, and can be grouped into two main approaches; policy gradient models and fully differentiable models. The first group of models makes use of reinforcement learning techniques like policy gradients (Williams, 1992) (see Section 2.3.4). In these models (Yu et al., 2017; Fedus et al., 2018), the discriminator network is disconnected from the generator network, while policy gradients are used to estimate a reward gradient from the discriminator back to the generator. The main issues with these

techniques is that they are susceptible to training difficulties of standard policy gradients methods including high variance and less sample inefficiency. In addition, they do not incorporate latent space learning, hence it does not allow learning higher representations of the data that can be later used to control the generation process to produce specific desired features. The other group of models employs a fully differentiable GAN network (Nie et al., 2019; Chen et al., 2018b; Zhang et al., 2017; Kusner and Hernández-Lobato, 2016). These models make use of Gumbel-Softmax trick (Jang et al., 2016b; Maddison et al., 2016b) to overcome the non-differentiability problem for discrete data, which allows the back-propagation from the discriminator back to the generator.

As we will explain later in Section 3.2.2, GAN-based models usually suffer from the "*mode-collapsing*" problem, which causes GAN to be incapable of generating diverse samples from the given latent codes. Although there were some empirical efforts to address this problem in some of these models (Chen et al., 2018b; Nie et al., 2019), the mode-collapsing issue has not been addressed yet in a principled way in these sequential models.

2.2.2.1 Controlled Sequence Generation

In many applications, we are not only interested in generating data similar to the real ones, but we need the outputs to have specific useful properties or attributes. For example, in molecular or drug design, certain desired useful properties in the outputs include high solubility and ease of synthesis (De Cao and Kipf, 2018; Guimaraes et al., 2017; Putin et al., 2018; Polykovskiy et al., 2018). In music generation, we might want the music to have specific pitch or tempo, or in text applications, the user might be interested in generating sentences according certain sentiment or tense (Hu et al., 2017). Therefore, generative models need to incorporate a mechanism to control the outputs of the generator towards achieving certain desired score or including useful properties. Several methods has been developed to solve this problem, where they can mainly categorized under two approaches: control by utilizing the latent variable space, or through incorporation of goal function optimisation.

Controlling through the Latent Space Latent space can be utilized as a controlling space. In such latent variable models, the learned space can learn meaningful hidden representations of the data (Radford et al., 2015). These representations might represent specific attributes learned from the data, like rotation(angle), age(young/old), emotional expression

or sentiment (sad/happy), or tense (past/present). This way, by navigating through the space of each learned attribute, an output with specific desired attributes can be generated.

The more this learned latent space is "*disentangled*" into distinct meaningful factors, the more the generation can be fully controlled. This happens when each factor (a dimension or group of dimensions of the latent variable) is responsible for a specific meaningful attribute of the data. A similar process is shown in Figure 2.5. However, this perfect disentangling is challenging to achieve, since often the learned variable is less perfectly disentangled along its dimensions, and might not be easily interpretable.

Controlling by Goal Optimisation Generation can also be controlled by optimizing desired user-defined objectives. This way, the model is encouraged to be trained to best achieve these desired objectives. For example, reinforcement learning techniques like policy gradients (Williams, 1992) can be incorporated into the model as an optimisation mechanism. This allows using objectives like BLEU (Papineni et al., 2002) score for text generation or molecular diversity reward (Putin et al., 2018) to encourage diverse molecules output in drug design.

However, most of the previous work address the controlled generation problem either through using RL or by learning leveraging latent space, but not both in a combined framework. This prohibits benefiting from both modes of control in a unified model. Recently, MolGAN (De Cao and Kipf, 2018) combined the benefits of latent space GAN with RL for drug molecular generation. Although chemical molecules can be represented as sequences, MolGAN processes them as graphs, not sequences, where the generator generates graphs of chemical molecules from input noise. In Chapter 4 we will describe this model in details and discuss the differences to our work regarding sequence generation.

2.3 Reinforcement Learning (RL)

Reinforcement learning is a general framework for solving sequential decision making problems. Most of RL problems in recent literature are mathematically formalized following the finite Markov Decision Processes (MDPs), which involves an agent interacting with an environment over time steps through actions, state transitions, and rewards feedback. In this section we introduce the key elements of MDPs: the returns, value functions, and Bellman equations, as well as the main approaches to solve RL problems under this formulation.

2.3.1 Finite Markov Decision Processes (MDPs)

MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision maker is called the ***agent***. The thing it interacts with, comprising everything outside the agent, is called the ***environment***. These interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent seeks to maximize over time through its choice of actions.

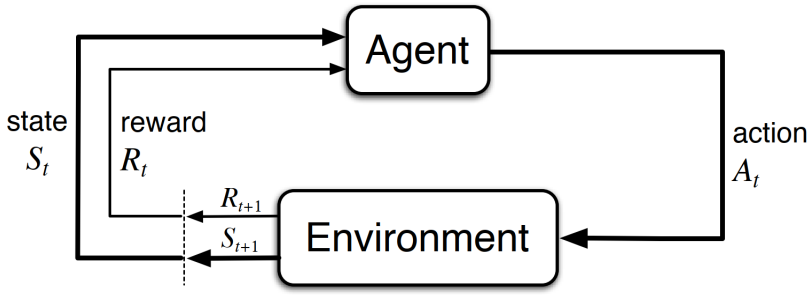

Figure 2.6: Agent-environment interaction in RL.

More specifically, the agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent receives some representation of the environment's state, $S_t \in \mathcal{S}$, and on that basis selects an action, $A_t \in \mathcal{A}$. One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R}$, and finds itself in a new state, S_{t+1}.

Thus the interaction sequence can be fully described by one ***episode*** (also known as "*trial*" or "*trajectory*") and the sequence ends at the terminal state S_T:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots, R_T, S_T.$$

This control setting was first proposed by (Bellman, 1957) and later extended to learning by (Barto et al., 1983). Comprehensive treatment of reinforcement learning fundamentals are provided by (Sutton and Barto, 2018).

Definition 10. A discrete time stochastic control process has the Markov property if :

- $p\left(S_{t+1} \mid S_t, A_t, \ldots, S_0, A_0\right) = p\left(S_{t+1} \mid S_t, A_t\right)$, and
- $p\left(R_t \mid S_t, A_t, \ldots, S_0, A_0\right) = p\left(R_t \mid S_t, A_t\right)$,

where $p(.)$ is a probability. All states in a MDP has "Markov" property, which means that the future of the process only depends on the current state, and the agent has no interest in looking at the full history. That is, the future and the past are conditionally independent given the present.

Definition 11 (Markov Decision Process (MDP))**.** A Markov Decision Process (Bellman, 1957) is a discrete time stochastic control process that is a 5 -tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $P : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the transition probability function $P\left(S_{t+1} = s' \mid S_t = s, A_t = a\right)$ (set of conditional transition probabilities between states), meaning the probability of transitioning into state s' if you start in state s and take action a,
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the reward function $R_t = R\left(S_t = s, A_t = a, S_{t+1} = s'\right)$, where $\mathcal{R}$ is a continuous set of possible rewards,
- $\boldsymbol{\gamma} \in [0, 1)$ is a discount factor.

The system is fully observable in an MDP, which means that what the agents observes from the environment is the state of the environment itself S_t. At each time step t, the probability

of moving to S_{t+1} is given by the state transition function $P(S_{t+1} \mid S_t, A_t)$ and the reward is given by a bounded reward function $R(S_t, A_t, S_{t+1}) \in \mathcal{R}$. This is illustrated in Figure 2.7.

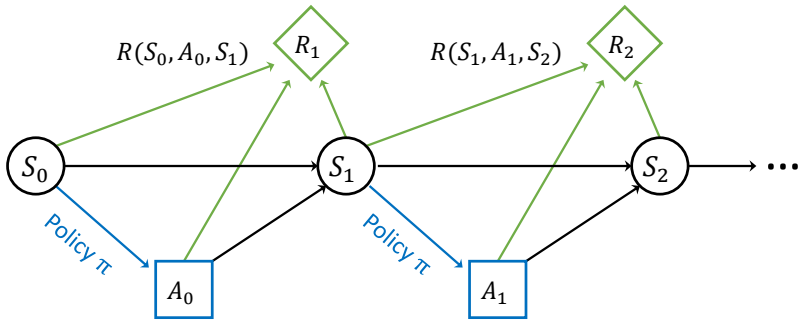

Figure 2.7: An illustration of a MDP. At each step, the agent takes an action that changes its state in the environment and provides a reward.

Definition 12 (Policy)**.** The policy is the agent's behavior function π, that tells us which action to take in state s. It is a mapping from state s to action a and can be either deterministic or stochastic. In the stochastic case it is defined as:

$$\pi(a|s) = P_\pi(A_t = a|S_t = s).$$

Definition 13 (Return)**.** The return is the total sum of discounted rewards going forward starting from time step t. Formally:

$$G_t \triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{2.3}$$

for all time steps $t < T$, even if termination occurs at $t+1$, where we define $G_T = 0$. In recursive form, G_t can be re-written as:

$$G_t = R_{t+1} + \gamma G_{t+1}. \tag{2.4}$$

The discounting factor $\gamma \in [0, 1)$ penalizes the rewards in the future. According to the nature of the problem at hand, the user might choose to give less or more weight to future rewards. This happens through tuning the value of γ.

2.3.2 Bellman Equations

Definition 14 (State-Value function V_π)**.** The ***state-value*** function V_π measures the goodness of a state or how rewarding a state is by a prediction of future rewards. The state-value of a state $S_t = s$ is the expected return at time t:

$$V_\pi(s) \triangleq \mathop{\mathbb{E}}_{\pi}[G_t \mid S_t = s] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \mid S_t = s\right], \text{ for all } s \in \mathcal{S}. \tag{2.5}$$

Definition 15 (State-Action function Q_π)**.** The ***state-action*** function Q_π measures the goodness of taking an action a while being in state s, or how rewarding is it to take action a in state s by a predicting future rewards. The ***state-action*** of an action $A_t = a$ and state $S_t = s$ is the expected return at time t:

$$Q_\pi(s,a) \triangleq \mathop{\mathbb{E}}_{\pi}[G_t \mid S_t = s, A_t = a] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \mid S_t = s, A_t = a\right]. \tag{2.6}$$

We can recover the state-value from the action-value using:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s,a). \tag{2.7}$$

Definition 16 (Bellman equation for V_π)**.** The Bellman equation for V_π is a recursive formula that expresses the relationship between the value of a state $V_\pi(s)$ and the values of its successor states $V_\pi(s')$ as:

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && \text{from 2.4} \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P(s', r \mid s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\right] \\
&= \sum_{a} \pi(a \mid s) \sum_{s', r} P(s', r \mid s, a)\left[r + \gamma V_\pi(s')\right] \\
&= \mathop{\mathbb{E}}_{\substack{a \sim \pi \\ s' \sim P}}\left[R_{t+1} + \gamma V_\pi(S_{t+1} = s') \mid S_t = s\right],
\end{aligned} \tag{2.8}$$

where s' denotes the state obtained or transitioned into from s after taking action a.

Definition 17 (Bellman equation for Q_π)**.** Similarly, the Bellman equation for a state-action

pair Q_π is defined as:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\
&= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P\left(s', r \mid s, a\right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s'\right]\right] \\
&= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P\left(s', r \mid s, a\right) \left[r + \gamma V_\pi\left(s'\right)\right] && \text{from 2.5} \\
&= \sum_{s',r} P\left(s', r \mid s, a\right) \left[r + \gamma \sum_{a'} \pi(a' \mid s')\, Q_\pi(s', a')\right] \\
&= \mathop{\mathbb{E}}_{s' \sim P} \left[R_{t+1} + \gamma \mathop{\mathbb{E}}_{a' \sim \pi} \left[Q_\pi(S_{t+1} = s', A_{t+1} = a')\right]\right]. && \text{from 2.7} \qquad (2.9)
\end{aligned}
$$

2.3.3 Main Approaches

Reinforcement learning systems differ in approaches to optimize and solve based on the available information in the MDP about the environment, or the ***model***. A model is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave (like predicting the next state and reward given the current state and action). Reinforcement learning spans the spectrum from low-level, trial-and-error learning to high-level, deliberative planning. Therefore, problems solved by RL systems mainly fall in two main categories:

- ***model-based:*** in which a model of the environment is known, and is used for planning, or deciding on a course of action by considering possible future situations before they are actually experienced.

- ***model-free***: where that model of the environment is not available, and the system cannot think about how their environments will change in response to a single action. Therefore, these methods are explicitly trial-and-error learners, as almost the opposite of planning. The tic-tac-toe player is model-free in this sense with respect to its opponent: it has no model of its opponent of any kind. Model-free methods rely on interaction with the environment to learn an approximate model of the environment by trial and error. For example, generating samples of state transitions and rewards, where these samples are then used to estimate state-action value functions. Because a model of the MDP is not known, the agent has to explore the MDP to obtain information.

This induces an exploration-exploitation trade-off which has to be balanced to obtain an optimal policy.

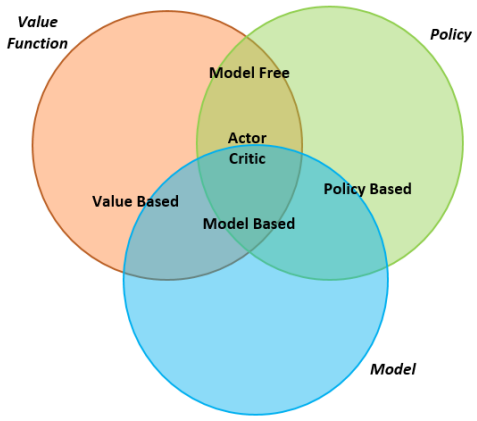

Figure 2.8: Relations between main RL methods

Because models have to be reasonably accurate to be useful, model-free methods can have advantages over more complex methods when the real bottleneck in solving a problem is the difficulty of constructing a sufficiently accurate environment model. Model-free methods are also important building blocks for model-based methods. Figure 2.8 illustrates a higher overview taxonomy of different RL methods. Although there are many algorithms that train model-free RL agents, there are mainly two categories:

- **Value Learning (Q-learning)**. Methods in this approach approximate the optimal action-value function $Q^*(s, a)$ by learning an approximation $Q_\theta(s, a)$. A classic example of this approach is Q-Learning and Deep Q-Learning (DQN) (Mnih et al., 2013), which uses a neural network to learn a approximate $Q_\theta(s, a)$ by optimizing parameters θ. This optimisation is usually performed off-policy, which means that each update can use trajectories collected by a different behavior policy, rather than that produced by the target policy. Q-learning methods indirectly optimize for the agent performance by learning a value Q_θ that is then used for obtaining the optimal target policy. There are many failure modes for this kind of learning, so it tends to be less stable. However, Q-learning methods are substantially more sample efficient, because they can reuse data more effectively than policy optimisation techniques.

- **Policy Optimisation**. In this approach, methods represent a policy explicitly as $\pi_\theta(a|s)$ where they optimize the parameters θ by gradient ascent on a performance objective $J(\pi_\theta)$. Examples include Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) and Proximal Policy Optimisation (PPO) (Schulman et al., 2017). The primary strength of policy optimisation methods is that they directly optimize the policy. This tends to make them stable and reliable, yet they tend to require more samples compared to value learning approaches.

Combined Policy Optimisation and Q-Learning. Recently, there has been growing interest in policy optimisation methods because of its explicit optimisation for the agent's policy. However, given the strengths and weaknesses of the two main approaches of RL, there can be a useful trade-off that carefully combine both approaches. An example of this category is Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015), which extends the discrete space DQN to continuous space within a policy optimisation framework.

Out of these different model-free approaches, we are mainly interested in the policy optimisation, since it is relevant and suitable for the kind of problems we address. In the following section, we describe the main method used in policy optimisation approach, named Policy Gradients (PG).

2.3.4 Policy Gradients

Policy gradients (PG) are the group of methods that solve RL problems by direct optimisation of the policy. This can be done by parameterizing the policy π with parameters θ and directly optimizing these parameters to maximize the expected future returns. The objective function is be defined as:

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[G_t(\tau)\right], \tag{2.10}$$

where τ is a trajectory, or an episode of T states and actions, and $G_t(\tau) = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ is the accumulative future reward over the trajectory at time step t. Then, we would like to optimize the policy π by gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta).$$

The gradient of policy performance, $\nabla_\theta J(\pi_\theta)$, is called the policy gradient. The proba-

bility of a trajectory $\tau = (S_0, A_0, ..., S_{T-1}, A_{T-1}, R_T)$ given that actions come from π_θ is:

$$p(\tau;\theta) = \rho_0(S_0) \prod_{t=0}^{T-1} P(S_{t+1} \mid S_t, A_t)\, \pi_\theta(A_t \mid S_t), \tag{2.11}$$

following the derivation in Appendix B, we find that the gradient ascent update can be computed by:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[G_t(\tau) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t \mid S_t) \right]. \tag{2.12}$$

More generally, the policy gradient update can be written as:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\Phi_t \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t \mid S_t) \right], \tag{2.13}$$

where Φ_t could be one of many function choices for exact or estimated future returns. In the next section, we discuss in details this general form and the different choices for Φ_t.

2.3.4.1 REINFORCE Algorithm

A well-known policy gradients algorithm is REINFORCE, which relies on estimating returns by Monte-Carlo methods (MC) to find the optimal policy π. The algorithm uses episode samples to update the policy parameter θ. Therefore, G_t is measured from real sample trajectories and used to update the policy π_θ using policy gradient Eq. (2.12). Algorithm 1 describes REINFORCE procedure.

Algorithm 1: REINFORCE algorithm

Initialize the policy parameters θ at random
Generate a trajectory (an episode) on policy
$\pi_\theta : S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$.
for *each step of episode* $t = 0, 1, \ldots, T-1$ **do**
  $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
  Update π_θ parameters by: $\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(A_t|S_t;\theta)$
end

2.3.5 Reducing Policy Gradients Variance

In practice, REINFORCE suffers from high variance for gradient estimates due to Monte Carlo sampling. As we will see later in Chapter 4, this variance affects the performance of deep learning models that are trained to learn the policy π to a specific task. By analyzing the variance of the policy gradients we find that it is proportional to the sum of cumulative returns of the chosen trajectories (please refer to (Takeshi, 2017) for more detailed derivation):

$$\begin{aligned}
\operatorname{Var}\left(\nabla_\theta J\left(\pi_\theta\right)\right) &= \operatorname{Var}\left(\mathbb{E}_\tau\left[G_t(\tau)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(A_t \mid S_t)\right]\right) \\
&= \operatorname{Var}\left(\mathbb{E}_\tau\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(A_t \mid S_t) G_t(\tau)\right]\right) \\
&\approx \sum_{t=0}^{T-1}\mathbb{E}_\tau\left[\Big(\nabla_\theta \log \pi_\theta(A_t \mid S_t) G_t(\tau)\Big)^2\right] \\
&\approx \sum_{t=0}^{T-1}\mathbb{E}_\tau\left[\Big(\nabla_\theta \log \pi_\theta(A_t \mid S_t)\Big)^2\right]\mathbb{E}_\tau\left[G_t(\tau)^2\right]. \qquad (2.14)
\end{aligned}$$

From Eq.(2.14) we find that the term $\mathbb{E}_\tau\left[G_t(\tau)^2\right]$ might cause an increase in the variance since it follows the randomness of the sampled trajectory τ from a stochastic policy π and the corresponding rewards. This means that if during training the samples trajectory yielded very highly positive or negative rewards, then the variance would increase.

One of the ways to reduce this variance is to subtract an arbitrary ***baseline*** $b(S_t)$ from the returns $G_t(\tau)$. The main intuition is that if an agent gets the rewards/return that it expected, it should "feel" neutral about it (getting a zero value in the $\big(G_t(\tau) - b(S_t)\big)$ term, thus zero gradient). Formally, this subtraction does not change the original policy gradients expectation since it is an unbiased estimate of the policy gradient.

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\left(G_t(\tau) - b(S_t)\right)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(A_t \mid S_t)\right]. \qquad (2.15)$$

Lemma 18. *The policy gradient with baseline $b(S_t)$ is an unbiased estimate of the standard policy gradient*

Proof. For all probability distributions:

$$\int_x \pi_\theta(x) = 1.$$

Take the gradient of both sides of the normalisation condition:

$$\nabla_\theta \int_x \pi_\theta(x) = \nabla_\theta 1 = 0.$$

Use the log derivative trick to get:

$$\begin{aligned} 0 = \nabla_\theta \int_x \pi_\theta(x) = \int_x \nabla_\theta \pi_\theta(x) = \int_x \pi_\theta(x) \nabla_\theta \log \pi_\theta(x) \\ \therefore 0 = \mathop{\mathbb{E}}_{x \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(x) \right]. \end{aligned}$$

Substituting x with $A_t \mid S_t$:

$$\mathop{\mathbb{E}}_{a_t \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(A_t \mid S_t) \right] = 0.$$

For any function b (which we call a ***baseline***) which only depends on state S_t:

$$\mathop{\mathbb{E}}_{a_t \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(A_t \mid S_t)\, b(S_t) \right] = 0. \tag{2.16}$$

From Eq. (2.12) and rearranging we can write:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\big(G_t(\tau) - b(S_t) \big) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t \mid S_t) \right].$$

Where we can see that the expectation for the last equation will still be equal to the original policy gradient expectation using Eq. (2.16). □

Similarly, the variance for the REINFORCE with baseline $b(S_t)$ can be approximated following (2.14) as:

$$\mathrm{Var}\left(\nabla_\theta J_{\text{baseline}}\left(\pi_\theta\right) \right) \approx \sum_{t=0}^{T-1} \mathbb{E}_\tau \left[\left(\nabla_\theta \log \pi_\theta(A_t \mid S_t) \right)^2 \right] \mathbb{E}_\tau \left[\left(G_t(\tau) - b(S_t) \right)^2 \right], \tag{2.17}$$

where we can see that the new term $\mathbb{E}_\tau \left[\left(G_t(\tau) - b(S_t) \right)^2 \right]$ allows for choosing a value for

$b(S_t)$ that can reduce the variance. For example, choosing $b(S_t)$ to be the expected value of $G_t(\tau)$ is the optimal choice. In fact, practitioners usually try to make $b(S_t) \approx \mathbb{E}\left[G_t(\tau)\right]$ to approximate the expected return starting at time t. As previously mentioned in (2.13), the general equation for REINFORCE can be generalized as :

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\Phi_t \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t \mid S_t)\right].$$

Under this general formulation, Φ_t might be one of the following functions:

1. $G_t(\tau)$, the cumulative rewards at time t
2. $G_t(\tau) - b(s_t)$, where $b(s_t)$ can be:
   (a) $\frac{1}{N}\sum_N G_t(\tau_N)$, the sample mean of N trajectory rollouts returns at time t.
   (b) $V_\pi(s_t)$, the state-value function at time t. In practice, V_π can be learned through a neural network V_ϕ that is trained concurrently with the policy network training π_θ.
3. $Q_\pi(s_t, a_t)$, the state-action value function.
4. $A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$, where $A_\pi(s_t, a_t)$ is called the advantage function. In this general form, the advantage function can be chosen among a family of estimators (Schulman et al., 2016) according to the method used to learn the state-value function.

2.4 Adversarial Examples

With the emergence of deep learning as an efficient learning technology for large data, a fundamental issue of this type of learning has been discovered, which is the deep learning models vulnerability to ***adversarial examples***. Adversarial examples are malicious inputs (e.g. images in the case of image classification model) that look like the original training data, but slightly perturbed to cause an imperceptible change from the human perspective. Researchers have found (Szegedy et al., 2013; Goodfellow et al., 2014b) that deep neural networks easily misclassify these perturbed images that look indistinguishable to "normal" images to the human eye. Consequently, the same phenomena has been found to be true for other types of input data including sequences or natural language. This behaviour has

important implications for safety and security if deep neural networks are to be deployed in the real world (e.g., in autonomous driving where images come from a camera sensor).

Formally, for a certain classification task, a deep neural network model f_θ is parameterized by θ. Given a data point x and a corresponding label y, we want to find an optimal θ so that $f_\theta(x)$ produces probabilities for all classes from its output layer, where the maximal component corresponds to the correct class index from y. Adversarial examples, denoted as $\tilde{x}$, can be written as $\tilde{x} = x + \eta$, where η is a small perturbation that causes an imperceptible change to the image, as judged by the human eye. Yet, despite the small perturbation, $f_\theta(\tilde{x})$ may behave very differently from $f_\theta(x)$, and place the highest output class probability to the wrong class.

2.4.1 Crafting an Imperceptible Attack

In order to craft an imperceptible attack, an appropriate value for η needs to be computed. According to (Goodfellow et al., 2014b), the appropriate value for η should be bounded by $\|\eta\|_\infty < \epsilon$, and $\epsilon > 0$ is small enough to be discarded by the data source (e.g. the smallest precision of possible image features in digital images). The effect of choosing η can be shown on a simple weight-input dot product:

$$\begin{aligned} w^\intercal \tilde{x} &= w^\intercal x + w^\intercal \eta \\ &= w^\intercal x + w^\intercal \epsilon \cdot sign(w), \end{aligned}$$

where w is a weight vector, $sign(w)$ is the sign (-1 or $+1$) of the weight vector w, and $\tilde{x} = x + \eta$ is an adversarial example. The main intuition here is that when using the perturbed input $x + \eta$ the weights of f_θ are grown by a value that should not cause a change to output class label on its own, but only with high input dimensions. By choosing $\eta = \epsilon \cdot sign(\cdot)$, (Goodfellow et al., 2014b) used this intuition to find a cheap and reliable method for generating adversarial examples called *Fast Gradient Sign Method* (FGSM):

1. Let $J(\theta, x, y)$ denote the cost (or loss) of training the neural network model.

2. The optimal perturbation η based on the gradients is $\eta = \epsilon \cdot sign\left(\nabla_x J\left(\theta, x, y\right)\right)$. Then

to produce an adversarial example $\tilde{x}$, use this equation:

$$\tilde{x} = x + \epsilon \cdot sign\left(\nabla_x J\left(\theta, x, y\right)\right).$$

FGSM was the first simple method to craft adversarial examples for image classifiers. Since then, more advanced and general methods where developed like *Projected Gradient Descent* (PGD) (Madry et al., 2018), where instead of computing the perturbation in only one step as FGSM, a maximisation procedure is used to find the best possible perturbation $\tilde{x}$ around point x that achieves the highest loss:

$$\max_{\|\tilde{x}-x\|_\infty \leq \epsilon} J\left(\theta, \tilde{x}\right).$$

Methods like FGSM and PGD are known as "white-box attacks", where they can "look-into" the model internals and have access to its parameters θ to compute the gradient with respect to input x. However, in a practical setting, the more probable scenario is that f_θ will be deployed as a trained black-box function. This means that other types of methods are needed to deal with these practical scenarios.

2.4.2 Types of Adversarial Attacks

In general, attacks using adversarial examples are either white-box or black-box attacks:

- **White-box attacks**. The attacker has access to the target model parameters θ, where computing the gradients of these parameters is possible, and are used to craft adversarial examples (Goodfellow et al., 2014b; Madry et al., 2018).

- **Black-box attacks**. The attacker has no (or little) information about the model parameters. In particular, it does not has access to the model parameters (Papernot et al., 2017). This disallows the computation of gradients of a loss function. However, the attacker can still ***query*** the model by providing the data to the model and then receiving its probability or label outputs. From there the attacker might decide to compute approximate or substitute gradients to craft the adversarial example.

Chapter 3

Adversarial Autoregressive Networks

Generating realistic sequences is a central task in many machine learning applications, including natural language generation and molecular and drug design. There has been considerable progress in developing deep generative models for sequence generation tasks, where a GAN framework is used to train the generative model. However, due to the underlying issue of "mode-collapse" in GANs, the lack of output diversity remains a main hurdle for the current models. Mode-collapsing happens when the GAN generator fails to learn many different modes of underlying data distribution, and only learns few data modes, leading to less-diverse outputs generated from only these modes. One of the reasons for this issue is the "mode-seeking" behaviour of the standard GAN objective. In this chapter, we propose a GAN-based generic framework to address the problem of mode-collapse. Specifically, we change the standard GAN objective to maximize the log-likelihood of data and minimize the Jensen-Shanon divergence between data and model distributions, simultaneously. This way, we encourage a balance between the "mode-seeking" behaviour of GAN and the "mode-covering" behavior of maximum likelihood estimation. In order to incorporate compressed latent representation in the generative model, we introduce a latent variable at the first token, and maximize its variational lower bound. We experiment our model with text generation task and show that it can generate realistic text with high diversity compared to the baseline. [1]

[1]The work in this chapter was published in the *Third Workshop on Bayesian Deep Learning, NeurIPS 2018* (Hossam et al., 2018).

3.1 Introduction

Recently, applying generative models like Generative Adversarial Networks (GAN) for sequence generation received an increasing attention. There has been a growing effort to develop GAN-based sequence generative models for different applications including natural language generation (Yu et al., 2017; Fedus et al., 2018) and molecular or drug generation (Guimaraes et al., 2017). Nonetheless, the underlying techniques used in these models cause specific training and generation problems.

Recent sequential GAN-based models can be grouped into two main approaches; reinforcement learning models and fully differentiable models. The first group of models makes use of reinforcement learning techniques like policy gradients (please refer to Section 2.3.4 for details). In these models (Yu et al., 2017; Fedus et al., 2018), the discriminator network is disconnected from the generator network, while policy gradients are used to estimate a reward gradient from the discriminator back to the generator. The main issues with these techniques is that they are susceptible to training difficulties of standard policy gradients methods including high variance and less sample efficiency. The other group of models employs a fully differentiable GAN network (Nie et al., 2019; Chen et al., 2018b; Zhang et al., 2017; Kusner and Hernández-Lobato, 2016). These models employ the Gumbel-Softmax trick (Jang et al., 2016b; Maddison et al., 2016b) to overcome the non-differentiability problem for discrete data, which allows backpropagating the learning gradient from the discriminator to the generator. Nonetheless, these models are susceptible to a well-known problem of GANs, which is lack of diverse outputs due to "mode-collapse".

Mode-collapse issue causes GANs to be incapable of generating diverse samples from the given latent values (as we will explain in details in Section 3.2.2). This issue happens when the generator fails to learn many different modes of underlying data distribution, and instead, is able to only learn few modes of data. Therefore, during inference time, the generator tends to generate samples from only those learned modes, ignoring other modes that were missed during training. There are different reasons that contribute to this problem (Goodfellow, 2016), including; *i)* the problem of catastrophic forgetting, a well-known problem of deep neural networks, where the network tends to forget previously learned tasks, *ii)* the GAN training procedure, and *iii)* the nature of GAN objective function used in practice.

In this chapter we address the problem of diversity and mode collapse from the angle of

GAN objective function. In details, we change the standard GAN objective to maximize the log-likelihood of data and minimize the Jensen-Shanon divergence between data and model distributions, simultaneously. This way, we encourage a balance between the "mode-seeking" behaviour of GAN and the "mode-covering" behavior of maximum likelihood estimation. Furthermore, in order to incorporate compressed latent representation in the generative model, we introduce a latent variable at the first token, and maximize its variational lower bound. Incorporating a latent space in the generative model allows learning high level representations from the input into a smaller more abstract latent space. This can be later helpful in controlling the generation process through changing/fixing specific factors of the latent space according to the desired outcome (see sections 2.2.1.2 and 2.2.2.1). While our work is demonstrated with discrete data, it can be straightforwardly adopted for continuous data.

In this chapter we propose a new framework called *Adversarial Auto-regressive Networks (ARN)*. The main highlights of our model include: (i) the capability of generating sentences from latent noise space; (ii) using standard back-propagation from with relaxation the discriminator instead of using reinforcement learning techniques with their associated problems; (iii) addressing the mode-collapse issue and achieving high diversity scores on natural language generation task.

In the following section we will present the technical background needed for this chapter, and in Section 3.3 we describe in details our proposed framework.

3.2 Background

3.2.1 Generative Adversarial Networks (GAN)

We briefly summarize here the basic idea of Generative Adversarial Networks (GAN) (Goodfellow et al., 2014a) (for full details, please refer to Section 2.2.1.2). GANs use adversarial training between two players to learn the probability density distribution of input data. The goal of the first player, the generator G, is to get good at generating data that is close to the real data distribution $p_d(x)$. The goal of the second player, the discriminator D, is to distinguish real data from fake data generated by the generator. Formally, the standard

GAN objective to optimize is the following minimax game between D and G:

$$\min_G \max_D \left(\mathop{\mathbb{E}}_{x \sim p_d} [\log D(x)] + \mathop{\mathbb{E}}_{z \sim p_z} [\log (1 - D(G(z)))] \right), \tag{3.1}$$

where z is the random noise input to G and p_z is the prior distribution of the z. After the training is finished, the generator is used to generate data from any random input z.

3.2.2 Mode Collapse

In practice, GANs usually suffer from "mode-collapse" problem (Metz et al., 2016; Poole et al., 2016; Hoang et al., 2018), where the generator learns to map several different noise z values to the same output point, relying on few modes learned from the true data distribution $p_{\text{data}}(x)$. This causes GANs to be incapable of generating diverse samples from the given input noise.

When mode collapse occurs in images for example, the generator makes multiple images that contain the same color or texture themes, or multiple images containing different views of the same object. The mode collapse problem is illustrated in Figure 3.1, where we see that rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one (Metz et al., 2016).

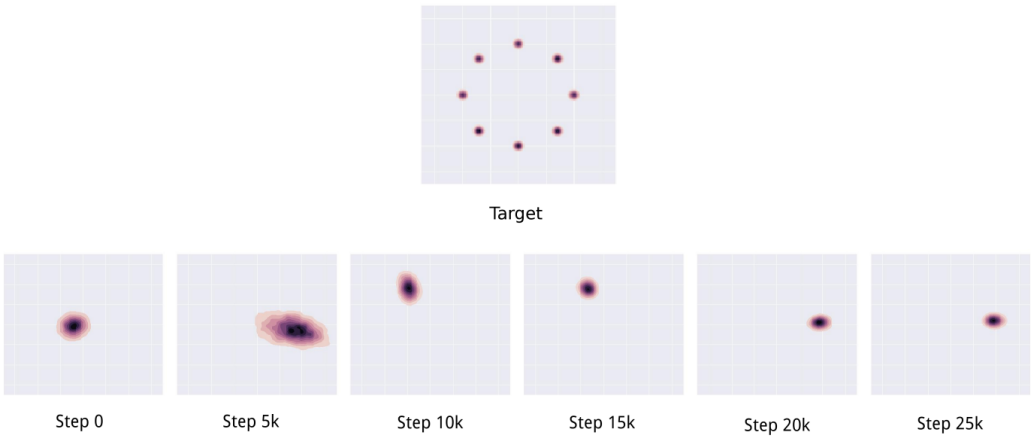

Figure 3.1: Mode collapse problem on a 2-D mixture of Gaussians data. The top row shows the true data distribution $p_{data}(x)$. The bottom row shows different distributions learned over epochs as the GAN is trained. (Metz et al., 2016)

One possible reason for this issue is that basic GAN objective resemble minimisation of reverse Kullback–Leibler divergence $RKL(P_{\text{data}} \parallel P_{\text{model}})$ between the data and model distributions. Minimizing reverse KL divergence has a mode-seeking behavior, in which it tries to minimize the divergence by seeking few modes of the data to match it. In contrast, the Kullback–Leibler divergence $KL(P_{\text{data}} \parallel P_{\text{model}})$ have mode-covering behavior, that try to optimize the model distribution in all areas of the data distribution, not only around few modes. Figure 3.2 shows the behavior of minimizing KL versus reverse KL divergences.

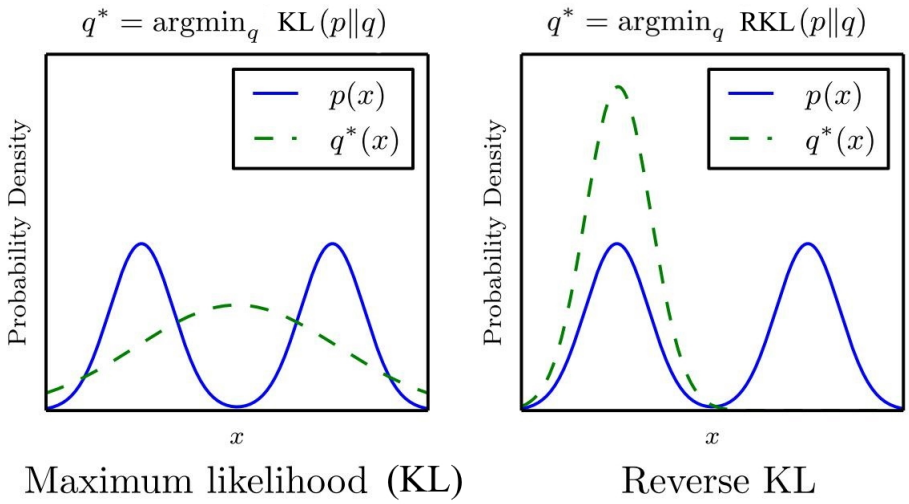

Figure 3.2: Behavior of minimizing KL divergence (left) and reverse KL divergence (right). (Goodfellow, 2016)

3.3 Proposed Framework

We describe in this section our proposed framework, Adversarial Autoregressive Networks (ARN) to overcome the mode collapse problem in sequence GANs. Adversarial Autoregressive Networks (ARN) consist of a recurrent autoregressive generator, like LSTM (Hochreiter and Schmidhuber, 1997) (see Section 2.2.1.1), trained in a GAN framework. We further incorporate a latent space that can be utilized in future applications to control the sequence generation by employing a variational autoencoder at the first token, x_1. Here, we derive and demonstrate the basic principles behind ARNs using recurrent autoregressive architectures,

but the same basic principles can be adapted for using other non-recurrent architectures. Figure 3.3 illustrates an overview of the main components of ARNs.

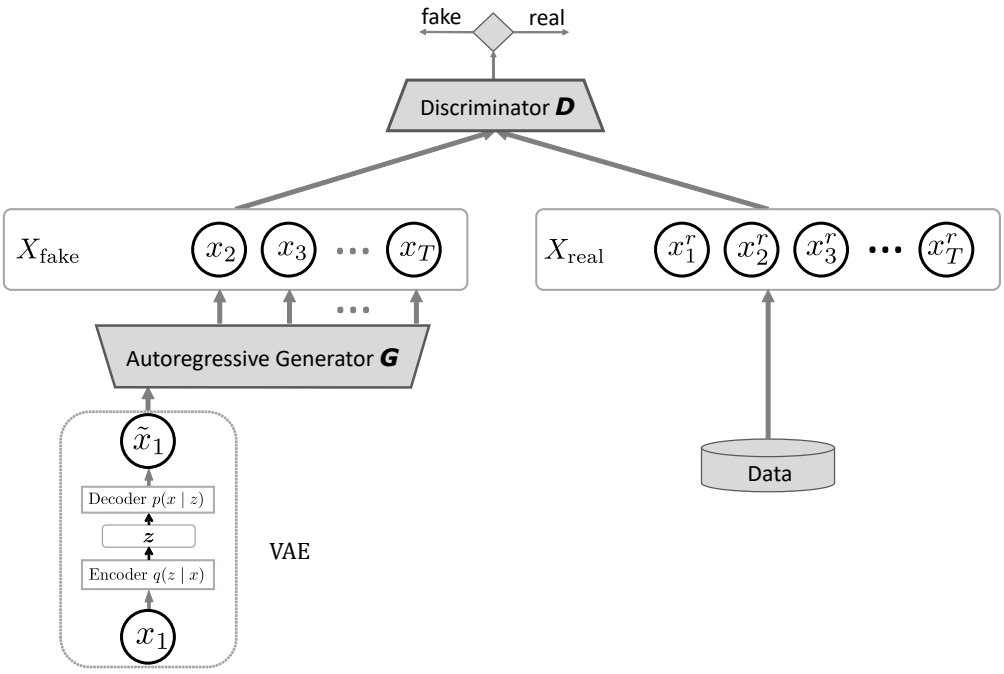

Figure 3.3: Proposed Adversarial Autoregressive Network

3.3.1 Model Definition

A sample X in our setting is defined as a sequence of T tokens denoted by $X = \{x_1, x_2, ..., x_T\}$, where we assume that all samples have length T. The general architecture of our proposed framework is shown in Figure 3.3, while that of the recurrent generator G is depicted in Figure 3.4.

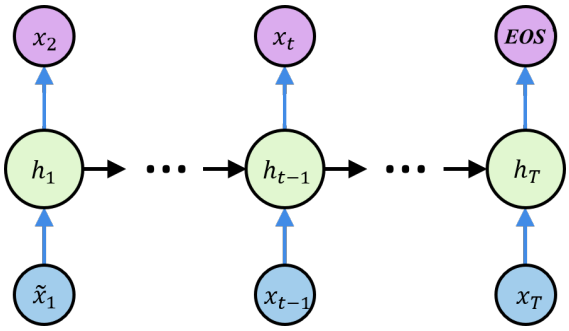

Figure 3.4: Training inputs and outputs for the recurrent generator G. The input is fit to a shifted output by one step, in order for G to learn to predict the next token given the current one. *EOS* is a special token for the end of the sequence.

For a recurrent autoregressive model G with parameters θ, the log-likelihood can be written as:

$$\log\, p_G(X \mid \theta) = \sum_{t=2}^{T} \log\, p_G(x_t|\, h_{t-1}, \theta) \,+\, \log\, p_G(x_1 \mid \theta). \tag{3.2}$$

This is the default neural recurrent model formulation. Now we start introducing an adversarial learning framework for this model by introducing a latent variable z to the model, where we rewrite $\log\, p(x_1 \mid \theta)$ as marginalisation over the z where KL is Kullback–Leibler divergence, $q(z \mid x_1, \phi)$ is an approximation of the posterior $p\,(z \mid x_1, \theta)$ and $p(z)$ is a prior distribution to z. We can then write $\log\, p_G(X \mid \theta)$ in terms of a lower bound (similar to original VAE (Kingma and Welling, 2013) derivation) as:

$$\begin{aligned}\log\, p_G(X \mid \theta) \geqslant \sum_{t=2}^{T} \log\, p_G(x_t|\, h_{t-1}, \theta) - KL(q(z \mid x_1, \phi) \,\|\, p(z)) \\ +\mathbb{E}_{q(z|x_1,\phi)}[\log\, p_G(x_1|\, z, \theta)],\end{aligned} \tag{3.3}$$

which has the same tightness of a VAE lower bound.

We propose to incorporate adversarial learning to the sequential model in a principled way. One generator $G\,(z)$ and one discriminator $D\,(X)$ are employed to create a game like in GAN while the task of the discriminator is to discriminate true data and fake data and the task of the generator is to generate fake data that maximally make the discriminator confused. In addition, the generator G is already available which departs from a noise $z \sim p_z$, uses the conditional distribution $p\,(x_1 \mid z, \theta)$ to generate x_1, and follows the autoregressive model to consecutively generate $x_{2:T}$. We come with the following minimax problem:

$$\max_G \min_D \left(\mathop{\mathbb{E}}_{X \sim p_d} [\log\, p_G\,(X \mid \theta)] - \mathop{\mathbb{E}}_{X \sim p_d} [\log\, D\,(X)] - \mathop{\mathbb{E}}_{z \sim p_z} [\log\,(1 - D\,(G\,(z)))] \right), \tag{3.4}$$

where the generator G consists of the decoder $p\,(x_1 \mid z, \theta)$, the autoregressive model, hence G is parameterized by (θ, ϕ), and $\log\, p_G\,(X \mid \theta)$ is substituted by its lower bound in Eq. (3.3). We can theoretically prove that the minimax problem in Eq. (3.4) is equivalent to the following optimisation problem:

$$\min_G\, KL\,(P_d \,||\, P_G) + JS\,(P_d \,||\, P_G)\,, \tag{3.5}$$

where JS is Jensen-Shannon divergence and P_G is the generative model distribution (see the proof in Appendix A). The optimisation problem in Eq. (3.5) reveals that at the Nash equilibrium point the generative distribution P_G is exactly the data distribution P_d, thus overcoming the mode-collapse issue caused by original GAN formulation.

To train our model, we alternatively update G and D with relevant terms. We note that in the optimisation for updating G regarding $\log p_G(X \mid \theta)$, we maximize its lower bound in Eq. (3.3) instead of the likelihood function.

3.3.2 Training

Likewise standard GAN models, to train ARN, we alternatively update the discriminator and generator using gradient ascent for these objective functions:

Update D:

$$\max_D \left(\mathop{\mathbb{E}}_{X \sim p_d} [\log D(X)] + \mathop{\mathbb{E}}_{z \sim p_z} [\log (1 - D(G(z)))] \right). \tag{3.6}$$

Update G:

$$\max_G \left(\mathop{\mathbb{E}}_{X \sim p_d} [\log p(X \mid \theta)] - \mathop{\mathbb{E}}_{z \sim p_z} [\log (1 - D(G(z)))] \right)$$
$$\text{or equivalently, } \max_G \left(\mathop{\mathbb{E}}_{X \sim p_d} [\log p(X \mid \theta)] + \mathop{\mathbb{E}}_{z \sim p_z} [\log D(G(z))] \right). \tag{3.7}$$

3.3.3 Relaxed argmax using Gumbel-Softmax

In order to overcome the non-differentiability of discrete data in our model, we use the Gumbel Softmax discrete relaxation (Jang et al., 2016a; Maddison et al., 2016a) between the variational decoder $p_g(x_1 \mid z)$ and the autoregressive network $p_G(x_t \mid h_{t-1}, \theta)$:

$$y_i = \frac{\exp((\log \pi_i + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log \pi_j + g_i)/\tau)} \qquad \text{for } i = 1, ..., k.$$

where y_i is the ith output of the Gumbel-Softmax layer, corresponding to the probability of the ith word in the vocabulary of size k. π_i and π_j are the un-normalized outputs from the last hidden layer, τ is the temperature parameter, and g_i and g_j both come from distribution Gumbel(0,1). We use temperature annealing schedule for τ from 1.0 to 0.5 as in (Jang et al., 2016a).

3.3.4 Overall Training and Inference

Putting all previous components together, ARN is trained in a similar fashion to a standard GAN, except with the diversity improved objective (Eq. 3.4) containing maximum log-likelihood through the generator $p_G(X_{2:T} \mid \cdot)$ and VAE encoder and decoder at x_1; $q(z \mid x_1)$ and $p_g(x_1 \mid z)$ respectively. For inference, the generator G is used to generate a sequence starting either from a normal noise $z_{noise} \sim N(0, I)$ or from a real first token code $z_{real} \sim N(\mu_\phi, \sigma_\phi \mid x_1)$. Therefore, it is easy to change any GAN sequence model into ARN framework, since it has the same end-to-end training fashion as GAN. In algorithms 2 and 3 we summarize the overall training and inference procedures of ARN.

Algorithm 2: ARN Training

Input: A sequence dataset $\mathcal{X} = \left\{X^1, X^2, ..., X^N\right\}$
Output: A trained generator $G_{\theta,\phi} = \{p_{G_\theta}, q_\phi\}$

Initialize p_{G_θ}, q_ϕ, and D_ψ with random parameters θ, ϕ, and ψ
for $i \leftarrow 1$ ***to*** *n_epochs* **do**
  Sample a batch of sequences $\mathcal{X}_B = \left\{X^1, X^2, ..., X^B\right\}$
  Update p_{G_θ} and q_ϕ via Eq. (3.7)
  Update D_ψ via Eq. (3.6)

Algorithm 3: ARN Inference

Input: A trained generator $G_{\theta,\phi} = \{p_{G_\theta}, q_\phi\}$
Output: A generated sequences batch $\left\{\tilde{X}^1, \tilde{X}^2, ..., \tilde{X}^B\right\}$

1 Sample a noise batch as $z_{noise} \sim N(0, I)$ or from a first token x_1 of a test-set sample $x_1 \in X^m_{\text{test}}$ as $z_{real} \sim N(\mu_\phi, \sigma_\phi \mid x_1)$
2 Get a first token batch $\tilde{x}_1 \sim p_{G_\theta}(x_1 \mid z)$
3 Recurrently (greedy) sample a batch of sequences $\tilde{x}_{2:T} \sim p_{G_\theta}(X_{2:T} \mid \tilde{x}_1)$
4 **return** $\left\{\tilde{X}^1 = \{\tilde{x}^1_1, \tilde{x}^1_{2:T}\}, \tilde{X}^2 = \{\tilde{x}^2_1, \tilde{x}^2_{2:T}\}, \ldots, \tilde{X}^B = \left\{\tilde{x}^B_1, \tilde{x}^B_{2:T}\right\}\right\}$

3.4 Experiments

To evaluate the performance of ARN with discrete sequential data, we apply the framework to natural language text generation task. We choose three common datasets: IMDB movie reviews dataset, MS-COCO, and EMNLP2017 WMT News. We then evaluate the performance through three different scores; *BLEU* score for text quality, *Diversity* score for text

diversity, and *FC* score for the percentage of the learned features of the overall true data features. The motivation is to measure the overall effectiveness in both the quality and diversity of our approach. Below we describe in details the implementation, evaluation scores, and discuss the experimental results.

Datasets and Implementation

- **The IMDB movie reviews dataset** (Maas et al., 2011b; Maas, 2011) consists of 50,000 reviews extracted from IMDB website. The training set is divided into two groups; positive, negative (12,500 reviews each), and the testing set is 12,500 positive and negative each. In all of our experiments, we use sentence length of 20 words and vocabulary size of 10,000. We use 500 dimensional embedding vectors, and 500 units hidden layer for LSTM Generator and 350 units for VAE.

- **The MS-COCO image captions dataset** (Chen et al., 2015) includes 3,826 unique words with the maximum sentence length of 10 words. Both the training and test data contain 7680 text sentences.

- **The EMNLP2017 WMT News dataset** (Guo et al., 2018) consists of 5,030 unique words with the maximum sentence length of 20, and we only use first 10,000 sentences from (Nie et al., 2019). Training set contains 6,988 sentences and test data contains 2,996 sentences.

Evaluation Metrics

We evaluate both the quality and diversity of the generated sentences. To evaluate quality, we use BLEU score (Papineni et al., 2002), which is commonly used in machine translation to compare the quality of candidate translations compared to the ground-truth reference. To evaluate the diversity, we use two n-gram based scores inspired by (Fedus et al., 2018), namely the *Diversity* and "*Feature Coverage* (*FC*)" scores.

BLEU Score for each sentence computes the ratio of n-grams generated from the model that matches with a true ground truth, called *reference* sentences and is defined as follows:

$$\text{BLEU} - \text{N} = \prod_{n=1}^{N} \left(\text{precision}_n = \frac{Count\,(\text{ Model generated n-grams} \cap X_{\text{test\_ref n-grams}})}{Count\,(\text{ Model generated n-grams})} \right)^{\text{weight}(n)},$$

where N is the final n-gram version of the score (usually from 2 to 5), and weight(n) is the weight for the current n-gram inside the product. Usually, weight$(n) = \frac{1}{N}, \forall n \in \{1, 2, \ldots, N\}$ for the cumulative BLEU score. However, in our setup, we used the individual BLEU score, where weight$(n) = \mathbb{1}\{n = N\}$.

Diversity Score We define the diversity as the ability to generate sentences with diverse n-grams that are not necessarily found in the test set. We measure this by computing the percentage of unique n-grams generated by the model relative to the number of all n-grams generated by the model. The score is defined for n-grams as follows:

$$\text{Diversity-n} = \frac{Count(\text{Unique model generated n-grams})}{Count(\text{ model generated n-grams})}.$$

Feature Coverage (FC) Score The FC score is used to measure how well the model covers all features (n-grams) of the data. The score is computed as the percentage of unique n-grams generated by the model that is found in the test set, relative to the number of all n-grams generated by the model. The score is defined for n-grams as follows:

$$\text{FC-n} = \frac{Count\,(\text{ Unique model generated N-grams} \cap X_{\text{test\_ref n-grams}})}{Count\,(\text{ model generated n-grams})}.$$

It is important to notice that FC score does not necessarily correlate with BLEU score, as it computes only the percentages of "unique" n-grams that match with the test set. This means that unlike BLEU score, FC score is affected by the diversity of the generated sentences, where the higher the unique n-grams count, the higher the FC score, and vice versa. Sentences can be generated from our model in two ways; by starting from a real first word through the decoder, or from noise input through z.

Baseline

We compare our model to SeqGAN (Yu et al., 2017), which is a well-known baseline for sequential generative models that uses a discriminator as a reward signal for training the generator in reinforcement learning framework. The main difference of our framework to SeqGAN is that the latter updates the generator using the reinforcement learning algorithm REINFORCE (Williams, 1992) (for full details on REINFORCE and reinforcement learning (RL), please refer to Section 2.3.4). This significantly increases the variance of the generator gradient updates and the final trained generator, which makes SeqGAN both slow to train

and produces high variance generator. Most of the other text generator models in the literature are based on SeqGAN's REINFORCE updates although with different variations (Fedus et al., 2018; Guo et al., 2018). Since we implement a simple text generation model based on proposed ARN framework, we therefore compare mainly with SeqGAN, where it is the original architecture that inspired and resembles the main characteristics of other REINFORCE-based models.

Table 3.1: BLEU, FC and Diversity scores for IMDB Reviews dataset

	BLEU-2 ↑	BLEU-3	FC-2 ↑	FC-3	Diversity-2 ↑	Diversity-3
SeqGAN	**84.48**	**51.97**	9.77	10.10	19.70	47.18
ARN (x_1 from data) (Ours)	81.30	46.74	**12.24**	**12.40**	**24.01**	**51.38**
ARN (x_1 from noise) (Ours)	81.04	46.19	**12.32**	**12.45**	**24.35**	**52.06**

Table 3.2: BLEU, FC and Diversity scores for MS-COCO dataset

	BLEU-2 ↑	BLEU-3	FC-2 ↑	FC-3	Diversity-2 ↑	Diversity-3
SeqGAN	**70.36**	**41.40**	2.25	2.37	5.80	11.84
ARN (x_1 from data) (Ours)	70.01	40.12	**4.01**	**4.29**	**15.22**	**28.79**
ARN (x_1 from noise) (Ours)	69.94	40.17	**4.01**	**4.28**	**15.38**	**29.08**

Table 3.3: BLEU, FC and Diversity scores for the small EMNLP dataset

	BLEU-2 ↑	BLEU-3	FC-2 ↑	FC-3	Diversity-2 ↑	Diversity-3
SeqGAN	60.52	20.11	5.61	3.29	19.03	42.41
ARN (x_1 from data) (Ours)	**61.38**	**21.82**	**5.76**	**3.68**	**23.71**	**49.56**
ARN (x_1 from noise) (Ours)	**60.75**	**21.30**	**5.77**	**3.65**	**23.95**	**49.68**

Discussion

In Tables 3.1, 3.2 and 3.3 we report 2-grams and 3-grams of each evaluation score on the three datasets. We can see that both ARN outperforms the baseline in both "Feature-Coverage" and the Diversity score. This suggests that our model is capable of learning the same overall features (n-grams) of the data, yet it is also able to generate more diverse samples out of these learned features. We show samples generated from ARN on both IMDB and MS-COCO datasets in tables 3.4 and 3.6, and from SeqGAN in Table 3.5. We can see that ARN samples are more diverse and do not suffer repetition like the samples from SeqGAN.

Table 3.4: Generated sample sentences of ARN trained on IMDB dataset

Our Model (starting from decoded x_1)
i had more interest for this movie . even though they rightly watched it once . it's neither entertaining nor
he just watched this show , one of the funniest moments of <UNK> or plot . i watched this and
victor is a zombie documentary of its own hype that <UNK> puzzles on many of my favorites . and every
i couldn't relate to this movie for the first time only because time i watched the director whose friend i
Our Model (starting from noise z)
first movie was a letdown of a film . the first 30 minutes of it was ok , until i
this movie is not only horrible as a bad movie , not because it has been based on history .
probably don't know i am a fan of horror movies , it was a little while ago and i think
another is one of the worst movies , ranking up there . the script is <UNK> by the film's predecessor
the first review that i can't figure out what's a <UNK> about this film when it came out in 1972
this most awaited movie and it seems to be the worst film ever . good plot , the storyline was
holy movie is just fun . it's not to do justice . the first monkey meets this hoping to be
this film grabbed the attention to the plot with standard <UNK> , and this never even saw a story with
what i saw this episode too , i thought it was awesome for several great actors . <br / science
although movie <UNK> if not to forget with a british tradition . i understand why the ideas presented well enough
this 1985 cult film " animal " was a <UNK> of a couple tells the most of youth , but
this disappointing . . . richard murray was a landmark in many two so far , such a pretentious crap
people misguided . i found way almost no battlestar <UNK> to complain about this movie (and like a <UNK>
this superbly finished the <UNK> , i was very excited about martin carter and scott he ran the tv series
this movie is really very sweet here . it reminded me of the dvd both brought us that devil's <UNK>
if just watched this movie when i was around 15 years ago and although it looks like it was boring

Table 3.5: Generated samples from SeqGAN on MS-COCO dataset. Mode-collapse occurred by repeating the sentences, showing how the model lacks output diversity *(best seen in color)*.

Sample sentences
the outside of a bus station is fairly empty .
man in motorcycle leathers standing next to street next to
woman with umbrella on a rainy day , near a
the outside of a bus station is fairly empty .
the outside of a bus station is fairly empty .
a person dressed as a giraffe carrying a bullhorn .
a his and her sink set in a bathroom .
woman with umbrella on a rainy day , near a
a computer desk with a cup of coffee , ,
a his and her sink set in a bathroom .
motorcyclists driving on a passing field people and some people
a computer desk with a cup of coffee , ,
woman with umbrella on a rainy day , near a
woman with umbrella on a rainy day , near a
a person dressed as a giraffe carrying a bullhorn .

Table 3.6: Low/average BLEU samples from ARN on MS-COCO dataset

BLEU-3	**Sample sentences**
00.00%	a woman putting on a lipstick holding a sharp knife
00.00%	a sleek kitchen includes walnut cabinets and shiny black counters
00.00%	a chinese street showing business signs hanging over the store
00.00%	people young women are washing two motorcycles with hoses .
00.00%	a modern flatscreen television sits upon a nostalgic console model
12.50%	several small kitchen contains a refrigerator and small counter .
12.50%	a blurry image with lights and cars in the background
12.50%	a bathroom wih a big mirror and a tv attached
25.00%	a colorful vegetable salad is in a green bowl .
25.00%	the bathroom has a pink tub and pink toilet .
25.00%	ta knife is standing up in a cutting board .
37.50%	a small efficiency apartment with a dark wooden table .
37.50%	a black cat is staring directly into the camera .
37.50%	this extravagant dessert on a plate overlooking the water .
50.00%	a herd of sheep passing by people behind barriers .
50.00%	a stainless steel toilet with a little pink bunny inside
50.00%	a black and silver motorcycle parked in the road .

To qualitatively evaluate the "quality" of this diverse output, we show generated samples for ARN on MS-COCO model with average or low BLEU scores in Table 3.6. We can see that low BLEU sentences generally still have grammatical structure and sometimes semantically meaningful, suggesting that the model does not learn random or gibberish modes when they do not match directly with data.

While the BLEU scores for the IMDB and MS-COCO are lower than the baseline, yet the *FC* and Diversity scores are higher. For text generation tasks, BLEU is computed relative to the whole test corpus. This means that BLEU score can increase significantly at the expense of output diversity, when few generated sentences match highly with test corpus but are repeated very frequently. Therefore, we see that it is essential that text generative models are evaluated for both quality and diversity in a unified manner (Fedus et al., 2018).

3.5 Summary

In this chapter we presented a sequence generative framework, Adversarial Auto-regressive Networks (ARN), to address the problem of low output diversity in GANs. The motivation behind this is that a useful sequence generative model needs to capture the underlying data

distribution accurately enough in order to generate diverse outputs.

To address the diversity issue in sequence GANs and alleviate the mode collapse problem, the proposed ARN framework combines GAN and MLE training. The motivation is to combine the standard GAN "mode-seeking" behaviour with MLE "mode-covering" behaviour. This way, the generator is encouraged to balance both behaviors, and trying to cover other modes of underlying data distribution, and not being trapped around only few modes. In details, we change the standard GAN objective to minimize KL divergence between data and model distributions, and minimize the Jensen-Shanon divergence, simultaneously.

In addition, we incorporated a compressed latent representation by introducing a latent variable at the first token, and maximizing its variational lower bound (Eq. 3.3). This can be helpful in future applications in controlling the generation process through the changing/fixing specific factors of the latent space according to the desired outcomes.

In order to evaluate our framework, we applied ARN for natural language generation task. Using three quality and diversity scores, we found that ARN generated grammatically and semantically meaningful sentences, and outperformed the chosen baseline SeqGAN, that depends on RL policy gradients in diversity and feature coverage scores:

- For diversity score, ARN outperforms SeqGAN by more than 18% difference (Table 3.2).

- In order to measure how close the learned diversity is to the data distribution (and not just incoherent randomness), we compute the *Feature-Coverage (FC)* score. ARN outperforms on *FC* score with absolute 2% scores higher than the baseline, thus demonstrating that learned diversity comes from the data distribution, and produces diverse and coherent sentences (Tables 3.4 and 3.6).

- We demonstrated ARN with discrete data, yet it can also be straightforwardly adapted for continuous data, as we show later in Chapter 4.

In conclusion, the proposed framework in this chapter represents our initial steps into developing useful generative models for sequence generation. By applying to text generation task, our experiments show that ARN can learn better diverse features from discrete input like text compared to the baseline. After addressing the diversity issue of GANs, we

look forward to incorporating critical features in ARN that enable real-world applications in different domains.

This can be achieved in two possible ways; *i)* leveraging the latent space in a fine controlled way to capture disentangled factors of variation from input distribution, thus allowing the user to control the outputs according to desired selected factors (such as controlling tense or sentiment of the sentences), and *ii)* add the ability to optimize the outputs to achieve desired attributes or properties (such as better domain expert score). In Chapter 4 we will discuss how to extend ARN to incorporate such optimisation capability in an end-to-end framework.

Chapter 4

Goal-Oriented Controlled Generation

Deep generative models opened the door for many potential real-world applications with its capacity to learn complex and high dimensional data distributions. Following the success of deep generative models with computer vision tasks, there has been growing interest in developing deep generative models for sequence generation tasks including natural language generation and molecular and drug design. However, current sequential generative models mainly generate sequences to closely mimic the training data, without much control on what the desired output from the model should be. Therefore, these models have been mainly used for simple or artistic applications. For other real-world applications, such as material or drug design, or autonomous motion planning, we are not only interested in generating data similar to the real ones, but we need them to have specific useful properties or attributes. The main challenge that hinders the application of generative models in such domains is the absence of mechanisms to optimize the generated outputs according to certain metrics or useful properties. In this chapter, we aim to address this key challenge, to create models that produce high quality sequences with higher diversity, and incorporate controlling mechanisms that allow controlled generation. We extend our previous model (ARN) (Section 3.3) by incorporating a reward signal feedback to train the model. This reward is intended to be chosen by the user to encourage model optimisation towards certain properties or attributes, according to the desired task of the model. We apply the proposed framework to two different tasks, and we show that our model is able to achieve higher desired properties than baselines, while not sacrificing output diversity. [1]

[1]The work in this chapter is published at *The 2020 International Joint Conference on Neural Networks (IJCNN 2020)* (Hossam et al., 2020)

4.1 Introduction

Learning to generate realistic sequences from existing data is essential to many artificial intelligence applications, including natural language generation, drug design, robotics, and music synthesis. In these applications, a generative model learns to generate sequences of different data types according to each task. For instance, natural language and speech are sequences of words or utterances, in robot motion planning, a trajectory is an action sequence learned from experiences or sensory data. Recently, there has been a growing interest in deep models for sequence generation following the success of Generative Adversarial Networks (GANs) (Goodfellow et al., 2014a) and Variational Autoencoders (VAEs) (Kingma and Welling, 2014) in image generation tasks (Radford et al., 2015; Gulrajani et al., 2017; Zhang et al., 2019; Dam et al., 2019; Razavi et al., 2019).

However, realizing the full potential of these models in aforementioned applications has many challenges, and one of these key challenges is the absence of mechanisms to optimize the generated outputs according to certain metrics or useful properties. Most of the current work on generative sequence models mainly learn to "resemble" the data, meaning to generate outputs that are close to the real distribution. However, in many applications, we are not only interested in generating data similar to the real ones, but we need them to have specific useful properties or attributes. For example, in drug design, useful properties include high solubility and ease of synthesis (De Cao and Kipf, 2018; Guimaraes et al., 2017; Putin et al., 2018; Polykovskiy et al., 2018). In music generation, we might want the music to have specific pitch or tempo, or in text applications, the user might be interested in generating sentences according certain sentiment or tense (Hu et al., 2017). Therefore, the lack of optimisation mechanisms in current models hinders their practical use in a wide range of real world applications.

In this chapter, we propose a new sequential generative framework, named OptiGAN[2], that can generate sequences resembling those in a given dataset and achieving optimal scores according to a desired goal (e.g., solubility and ease of synthesis in drug design). Our proposed framework leverage GAN for mimicking real data and policy gradient reinforcement learning (RL) for optimizing a score of interest. It is well-known that although GANs can resemble real data, they face the mode collapsing problem (Goodfellow, 2016; Nguyen

[2]Our code is available here : `https://github.com/mahossam/OptiGAN`

et al., 2017; Hoang et al., 2018; Le et al., 2018b; 2019), hence leading to generate less diverse examples. We adopt the same solution we presented in Chapter 3 to this problem, by combining maximum likelihood and GANs (see ARN model in Section 3.3). We then leverage policy gradient RL into our model for optimizing a score of interest according to a desired goal (see Section 4.4). When incorporating policy gradient RL to ARN framework we observe that the variance of gradient estimation is very high, hence leading to unstable training. To resolve this issue, we resort to the Monte Carlo rollout in (Yu et al., 2017) with a slight modification (see Section 4.4).

We demonstrate the capacity of OptiGAN in two applications: text generation (discrete data) and air combat trajectory generation (real-valued data). For text generation task, we aim to generate sentences resembling real sentences in a given text corpus, while optimizing the BLEU (Papineni et al., 2002) score for obtaining better quality natural language sentences. For aircraft trajectory generation task, we aim to generate a trajectory plan for air-combat maneuver scenario between two aircraft and optimize the McGrew score (McGrew et al., 2010) which reflects the tactical quality of aircraft trajectories in an air combat (McGrew et al., 2010). In both applications, we show that we can generate high quality outputs and achieve higher scores than current related models aided by the RL component, while preserving the diversity of generated outputs using our hybrid maximum likelihood GAN.

The main contributions include:

- We propose OptiGAN which has the following advantages: (i) *an end-to-end generative framework with incorporated goal optimisation mechanism,* (ii) *general formulation that can be used for wide variety of different goals and models*, and (iii) *optimizing for desired goals without sacrificing output sample diversity.*

- To the best of our knowledge, OptiGAN is the first GAN controlled generative model for sequences that addresses the diversity issue in a principled approach. OptiGAN combines the benefits of GAN and RL policy learning, while avoiding their mode-collapse and high variance drawbacks.

- We investigate the problems of interest comprehensively and our findings would advance the understanding of the behavior when incorporating GAN-based models with RL. Specifically, we empirically show that if only pure RL is applied to maximize a

score of interest with the GAN-based objective, realistic quality of the output might be sacrificed for the sake of superficially obtaining higher scores. For instance, in the case of text generation, the model was able to cheat the selected quality score by generating sentences in which few words are repeated all the time. This shows that combining a GAN-based objective with RL encourages the optimisation process of RL to stay close to the true data distribution.

4.2 Related work

In this section we summarize and discuss the literature review in Section 2.2.2.1 on controlled sequence generation. Since we mainly address the controlled generation problem in this chapter, we refer the reader for Section 2.2.2 and Chapter 3 for more general literature review on sequence generative models.

Controlled sequence generation could be mainly achieved through two mechanisms; by utilizing the latent variable space (Engel et al., 2019; Polykovskiy et al., 2018), or through incorporation of desired goal optimisation (De Cao and Kipf, 2018; Putin et al., 2018). For latent space models like VAE and GAN, the latent space can be utilized as a fine control of the generated outputs. The benefit of such models is that they can learn meaningful compressed representations of the data (Radford et al., 2015). These representations might represent specific attributes learned from the data, like rotation(angle), style, age(young/old), or emotional expression (sad/happy) in case of images. This way, by navigating through the space of each learned attribute, an output with specific desired attributes can be generated.

Generation can also be controlled by optimizing desired user-defined objectives, where the generative model is designed in such a way, that it optimizes certain environment evaluation scores. Reinforcement learning (RL) techniques have been incorporated into generative models to enable such optimisation mechanism. For example, VAE and GAN based models have been developed for goal optimized drug design (De Cao and Kipf, 2018; Putin et al., 2018; Polykovskiy et al., 2018), where VAE latent space or GAN augmented with reinforcement learning are used for to maximize drug design scores like druglikeliness, synthesizability, or solubility. However, as mentioned in Section 2.2.2.1, most of the previous work address the controlled generation problem either through using RL or by leveraging latent space, but not both in an integrated framework.

Recently, MolGAN (De Cao and Kipf, 2018) combined the benefits of latent space GAN with RL for drug molecular generation. The model is developed for graph generation, where the generator generates graphs of chemical molecules from input noise, and a convolutional discriminator network is used for the adversarial training. In addition, the model utilizes an additional reward network to optimize the learning according to molecular rewards received from external chemical software. However, the model is developed for graph generation, not for sequences, as the input graphs are processed in its entirety and the model does not learn probability dependencies in an autoregressive manner from its graph inputs. Therefore, the neural architecture used for MolGAN's generator is multi layer perceptrons (MLP), which was used for simplicity and ease of training as compared to recurrent layers. A key difference between our proposed framework and MolGAN is that we develop a framework for sequence learning, where we use a recurrent autoregressive network for the generator, and model the inputs as sequence of states and actions. Because of it represents inputs as entire graphs, MolGAN lacks this sequential architecture, and uses a simplified variant of Deep Deterministic Policy Gradient (DDPG) (please refer to Section 2.3.3) that does not model the inputs as state-action sequential process.

4.3 Background

Below we revise basic background and concepts needed for our proposed framework; Adversarial Autoregressive Networks (ARN) and policy gradients.

Adversarial Autoregressive Networks (ARN)

As previous described in Section 3.3, Adversarial Autoregressive Networks (ARN) is a sequential generative framework that addresses mode-collapse problem in GANs through a modification in of the standard GAN training objective. In ARN, the standard GAN objective is changed to both maximize the log-likelihood of data and minimize the Jensen-Shanon divergence between data and model distributions, simultaneously. For a sample X defined as a sequence of T tokens denoted by $X = [x_1, x_2, ..., x_T]$, ARN optimizes the following minimax problem:

$$\max_{G} \min_{D} \left(\mathbb{E}_{X \sim p_d} \left[\log \, p_G \left(X \mid \theta \right) \right] - \mathbb{E}_{X \sim p_d} \left[\log \, D \left(X \right) \right] - \mathbb{E}_{z \sim p_z} \left[\log \left[1 - D \left(G \left(z \right) \right) \right] \right] \right). \quad (4.1)$$

Reinforcement learning using Policy Gradients

Reinforcement learning is a general framework for sequential decision making that learns an agent policy to maximize rewards from the environment (for full details, please refer to Section 2.3).Policy gradients (Sutton and Barto, 2018) (please refer to Section 2.3.4) is a well-known reinforcement learning policy learning technique, that explicitly learns the optimal agents policy. The objective is to maximize the expected future returns over an episode of T time steps:

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[U_t(\tau) \right],$$

where π is the "policy" to be optimized, θ are the parameters of π, and U_t specifies the cumulative reward of an episode at time t (please notice that here we use U as another symbol for the returns to avoid confusion with the generator symbol G). The model is then updated via gradient ascent with this gradient estimate:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi \left[\sum_t U_t \nabla_\theta \log \pi \left(A_t \mid S_t, \theta \right) \right], \quad (4.2)$$

where A_t is an action chosen at time step t by the agent's policy π given the current state of the environment S_t.

4.3.1 Problems of Interest

We demonstrate the capacity of our proposed framework in two applications of interest: *text generation* and *air combat trajectory generation*. For each application, our task is to generate sequences that achieve two concurrent goals: i) mimicking those in a given dataset and ii) obtaining high scores specified by an optimized goal which might be varied for specific tasks.

Text generation

We need to generate sentences that are similar to real sentences in a given text corpus and have high quality from human justification. A well-known score used to measure the quality of generated sentences is BLEU score (Papineni et al., 2002). Specifically, the BLEU score for each sentence computes the ratio of n-grams generated from the model that matches with

a true ground truth, called *reference* sentences and is defined as follows:

$$\text{BLEU-N} = \prod_{n=1}^{N} \left(\text{precision}_n = \frac{Count(\text{ Model generated n-grams} \cap X_{\text{test\_ref n-grams}})}{Count(\text{ Model generated n-grams})} \right)^{\text{weight}(n)},$$

where weight(n) is the weight for the current n-gram inside the product. We used the common cumulative weights where weight$(n) = \frac{1}{N}, \forall n \in \{1, 2, \ldots, N\}$. In our proposed model, beside generating realistic sentences, we also aim to maximize the BLEU score of generated sentences. As shown later, we utilize the BLEU score as reward function in our RL inspired framework.

Air combat trajectory generation

For air combat missions, pilots are trained to conduct certain maneuvers according the combat situation they face. There are well known maneuvers that the pilots are trained on, either defensive, offensive, or neutral. However, modeling the maneuvers through generative models could help pilots to simulate many different or novel trajectories that were not considered before. We consider a specific air combat maneuver between two fighters called "*Stern Conversion*" maneuver (Austin et al., 1990). In this maneuver, the opponent (the red aircraft) flies in a straight and level line, and does not detect the blue aircraft, while the blue aircraft, on the other hand, tries to get behind the opponent aircraft, in order to increase the chance to engage it (see Fig. 4.1). The goal of the generative model in this context is to train a generator to generate a blue aircraft trajectory given the red aircraft trajectory.

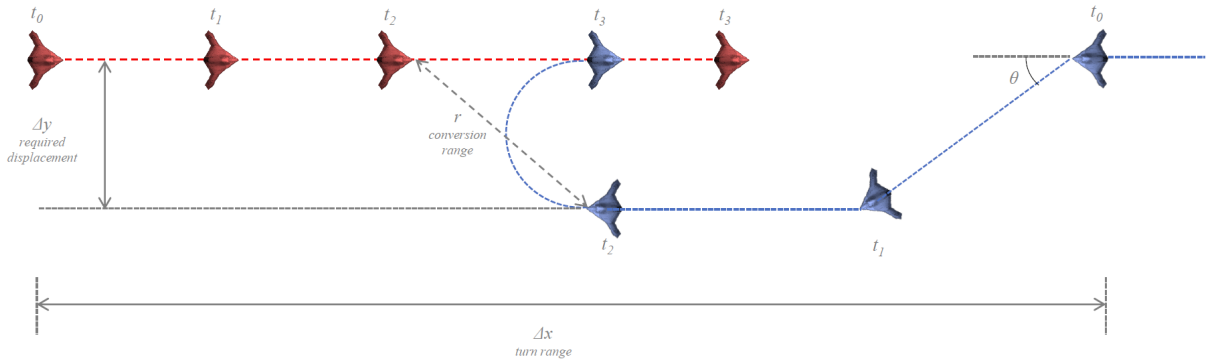

Figure 4.1: "Stern Conversion" Flight Maneuver.

In addition to generating realistic trajectories, we also need to maximize the McGrew

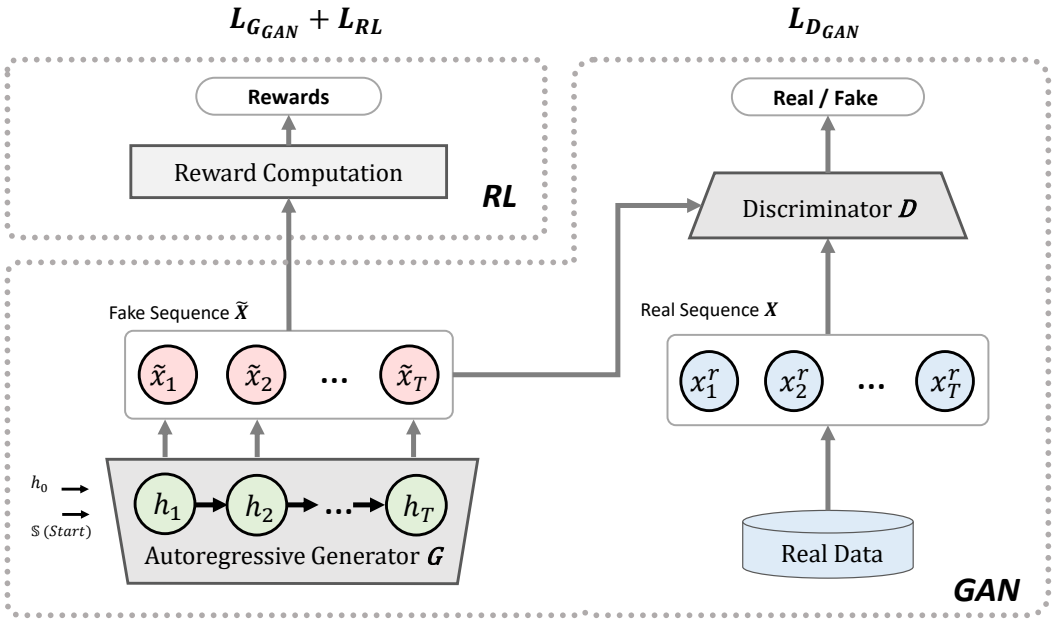

Figure 4.2: Overview of OptiGAN framework. The Reinforcement Learning (RL) component is incorporated with sequence GAN model. The generator G is trained by combining two losses, the GAN loss and the RL loss, $L_{G_{GAN}}$ and L_{RL}.

score (McGrew et al., 2010), which measures the score of how well the blue aircraft was doing relative to the red aircraft in an attempt to get behind the red aircraft (please refer to Appendix C and (McGrew et al., 2010) for more details). Due to security restrictions, we cannot access the real trajectories sensory data. Instead, we use *ACE-Zero* (Ramirez et al., 2017) air combat flight simulator to generate the training data. This *simulator* was developed by domain experts to imitate the real aircraft trajectories. As demonstrated in the experiments section, our model can generate novel trajectories with high McGrew score close to the average scores for ACE-Zero trajectories.

4.4 Proposed Framework

Starting from the ARN framework (described in details in Section 3.3), we incorporate an additional reward signal that allows the model to maximize desired scores/goals according to the current task. Specifically, we incorporate policy gradients RL that allows us to train our model end-to-end. Finally, to stabilize the training process, we apply variance reduction techniques when training with policy gradients. The final model is named OptiGAN whose overview architecture is shown in Fig. 4.2.

In details, to incorporate the ability to model the data to maximize rewards from the environment, we use policy gradients to learn a policy that maximizes the total rewards from environment. As mentioned in Section 4.3, the learning objective to maximize the return rewards over an episode from $t = [0, 1, ..., T-1]$ is:

$$J(\pi_\theta) = \mathbb{E}_\pi \left[\sum_t U_t \log \pi \left(A_t | S_t, \theta \right) \right].$$

In our model, the policy is the generator G, and the state at time t is the hidden state of the generator h_t. Thus the objective becomes:

$$J(\pi_\theta) = \mathbb{E}_{X \sim P_G} \left[\sum_t U_t \log p_G \left(X \mid h_t, \theta \right) \right].$$

We then use REINFORCE (Williams, 1992) algorithm (see Section 2.3.4.1) to find the optimum parameters for policy G by gradient ascent of the gradient of J as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{X \sim P_G} \left[\sum_t U_t \nabla_\theta \log p_G \left(X \mid h_t, \theta \right) \right].$$

The last equation is the general formulation for any task we train OptiGAN for. For each specific task, the returns U_t is chosen appropriately according to the nature of the task. For instance, for the text generation task, we use the BLEU score as reward value, and for the air combat trajectory generation task, we use the McGrew score as reward value.

Training procedure. To train an OptiGAN model, we alternatively update the discriminator and generator using gradient ascent for these objective functions:

- Update D:

$$\max_D \mathbb{E}_{X \sim p_d} \left[\log D \left(X \right) \right] + \mathbb{E}_{z \sim p_z} \left[\log \left[1 - D \left(G \left(z \right) \right) \right] \right].$$

- Update G:

$$\max_G \left(\mathbb{E}_{X \sim p_d} \left[\log p \left(X \mid \theta \right) \right] - \mathbb{E}_{z \sim p_z} \left[\log \left[1 - D \left(G \left(z \mid \theta \right) \right) \right] \right] - \mathbb{E}_{X \sim P_G} \left[\sum_t U_t \log p_G \left(X \mid h_t, \theta \right) \right] \right). \tag{4.3}$$

It is worth noting that for discrete data (e.g. text), we define the likelihood $p_G(x_i \mid h_i) = \text{softmax}(W_o h_i)$ where W_o is the output weight matrix. In addition, to allow end-to-end

training, we apply Gumbel softmax relaxation (Maddison et al., 2016b; Jang et al., 2016b) (see Section 3.3.3) for the discrete case. We fix a special start token to $p(x_1|z) = 0$, as we depend on Gumbel Softmax for random output sampling. For real-valued data (e.g., air combat trajectory), we employ $p_G(x_i \mid h_i) = \mathcal{N}(W_o h_i, \sigma^2)$ where σ is the standard deviation parameter.

Final generator loss. Following Eq. 4.3, the final loss function to train the generator G with adversarial training and policy gradients is:

$$\max_G \left(\omega \, \mathbb{E}_{X \sim p_d} \log p\left(X \mid \theta\right) + \lambda \, \mathbb{E}_{z \sim p_z} \log D\left(G\left(z\right)\right) + \alpha \, \mathbb{E}_{X \sim P_G} \left[\sum_t U_t \log p_G\left(X \mid h_t, \theta\right) \right] \right),$$

where ω, λ and α are hyper-parameters that control how much the effect of log-likelihood, adversarial training and policy gradients are on the total loss.

Reducing policy gradients variance

To reduce the variance of the policy gradients and to help policy gradients converge faster toward optimal solution, we use policy gradients with baseline (Sutton and Barto, 2018) (Section 4.4), where the policy gradient is defined as:

$$\begin{aligned} \nabla J(\theta) &= \mathbb{E}_\pi \left[\sum_t \left(U_t - b\left(S_t\right)\right) \nabla_\theta \log \pi\left(A_t \mid S_t, \theta\right) \right] \\ &= \mathbb{E}_{X \sim P_G} \left[\sum_t \left(U_t - b\left(h_t\right)\right) \nabla_\theta \log p_G\left(X \mid h_t, \theta\right) \right], \end{aligned}$$

where $b\left(S_t\right)$ is a baseline, a function that can be estimated or learned during training. The use of a baseline does not change the gradient expected value, but in practice, reduces its variance. In our experiments, $b(S_t)$ is a fixed value equivalent to the average of computed rewards over training time.

In addition, in order to further reduce the variance of policy gradients, we use an algorithm similar to the Monte Carlo rollout in (Yu et al., 2017) with a slight modification. We generate few complete sentences at each time step t onward, and take their average as U_t at that time step. The simple change in our case is that instead of getting the reward value from a discriminator as in (Yu et al., 2017), we directly compute the reward according to the

chosen score (e.g. the sum of all n-gram BLEU scores in text generation case). The detailed algorithm for this procedure is described in Algorithm 4.

Algorithm 4: OptiGAN Monte Carlo rollout to reduce REINFORCE variance

Initialize the policy parameters θ at random
Generate a trajectory (an episode) on policy
$\pi_\theta : S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$.
for *each step of episode* $t = 0, 1, \ldots, T-1$ **do**
  for *rollouts* $n = t+1, t+2, \ldots, T-1$ **do**
    Sample a sequence rollout $\tilde{X}_{t+1}$ starting from time step n as $\tilde{X}_{t+1} \sim G^T_{t=n}$
    $U_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k(\tilde{X}_{t+1})$
    Update π_θ parameters by: $\theta \leftarrow \theta + \alpha U_t \nabla_\theta \log \pi_\theta(A_t|S_t;\theta)$
  end
end

4.5 Experiments

We describe in this sections the experiments conducted on the two tasks in Section 4.3.1; Text and air-combat trajectory generation. First we describe the baselines used for both tasks, then we separately describe datasets, evaluation metrics, and discuss results for each task.

Baselines

We evaluate our proposed model for both discrete (in our case, text generation) and real-valued data (air-craft trajectory generation), summarized in Table 4.1.

Table 4.1: Comparison Baselines

	Discrete Data (Text)	Real-valued Data (Trajectories)
SeqGAN*	✓	–
LSTM	–	✓
OptiGAN-OnlyRL	✓	–
OptiGAN-OnlyGAN	✓	✓
OptiGAN	✓	✓

** SeqGAN works only with discrete data, not real-valued data*

First, for text generation, we compare with three baselines:

1. **SeqGAN** (Yu et al., 2017): is a well-known baseline for sequential generative models that uses a discriminator as a reward signal for training the generator in reinforcement learning framework.

2. **OptiGAN-OnlyRL:** This model is the vanilla reinforcement learning using policy gradients. For fairness, we implement it by using our own model with GAN component canceled, by zeroing out the GAN loss part.

3. **OptiGAN-OnlyGAN**: The sequence GAN with LSTM generator and discrete relaxation nodes, without any policy gradient component. We implement it using our model with RL component canceled, by zeroing out the policy gradient loss.

The GAN network implementation of our model is based on RelGAN with same hyperparamters and temperature scheduling, but using LSTM unit instead of relational memory. Finally, for trajectory generation, we implement two different models to compare with; LSTM (The MLE component of our model without adversarial training) and OptiGAN-OnlyGAN (our model without RL component), and we conduct ablation study for the effect of the RL component.

4.5.1 Text generation

Below we describe in details the experiments for OptiGAN with discrete data on text generation task.

Evaluation Metrics

We use both BLEU score and negative log-likelihood (NLL) mentioned below to evaluate the quality and diversity of our model.

BLEU Score As discussed in Section 4.3.1, BLEU score (Papineni et al., 2002) is a well-known text quality score in machine translation and text generation tasks. The higher the BLEU score is, the more the number of matching n-grams with the test set. In practice, and as discussed later, the BLEU score can be easily cheated by repeating few matching n-grams in one sentence, or by generating only one or a few high quality sentences from the model after training. This situation implies low output quality or diversity from the model.

Table 4.2: BLEU scores and NLL values on MS-COCO dataset

	BLEU-2 ↑	BLEU-3	BLEU-4	BLEU-5	NLL ↓
SeqGAN	75.09 ± 0.84	51.58 ± 1.06	32.06 ± 0.98	20.03 ± 0.68	0.830 ± 0.176
OptiGAN-OnlyRL	79.23 ± 3.76	59.23 ± 6.21	40.65 ± 7.15	27.11 ± 6.36	0.803 ± 0.106
OptiGAN-OnlyGAN	75.96 ± 0.71	53.79 ± 0.99	34.34 ± 0.86	21.51 ± 0.56	$\mathbf{0.735 \pm 0.080}$
OptiGAN (RL+GAN)	$\mathbf{76.55 \pm 0.72}$	54.28 ± 1.01	34.84 ± 0.86	22.06 ± 0.59	0.739 ± 0.080
OptiGAN (MLE+RL+GAN)	76.42 ± 0.70	$\mathbf{54.40 \pm 0.99}$	$\mathbf{35.06 \pm 0.90}$	$\mathbf{22.25 \pm 0.66}$	0.737 ± 0.082

Table 4.3: BLEU scores and NLL values on EMNLP News 2017 Dataset

	BLEU-2 ↑	BLEU-3	BLEU-4	BLEU-5	NLL ↓
SeqGAN	$\mathbf{76.05 \pm 1.67}$	47.60 ± 1.51	23.88 ± 0.88	12.05 ± 0.40	2.359 ± 0.272
OptiGAN-OnlyRL	79.16 ± 2.18	53.57 ± 4.00	31.26 ± 4.66	16.67 ± 3.14	2.267 ± 0.154
OptiGAN-OnlyGAN	73.15 ± 2.35	48.00 ± 1.32	26.12 ± 1.00	13.77 ± 0.71	2.234 ± 0.152
OptiGAN (RL+GAN)	74.32 ± 1.96	$\mathbf{48.93 \pm 1.20}$	$\mathbf{26.99 \pm 0.95}$	$\mathbf{14.47 \pm 0.66}$	$\mathbf{2.225 \pm 0.134}$
OptiGAN (MLE+RL+GAN)	74.03 ± 1.69	48.73 ± 1.08	26.64 ± 1.07	14.05 ± 0.79	2.226 ± 0.148

Negative Log-Likelihood (NLL) We use the negative log-likelihood of the generator (Nie et al., 2019) to measure diversity, defined as:

$$\mathrm{NLL}_{\mathrm{gen}} = -\mathbb{E}_{x_{1:T} \sim P_d} \log P_{G_\theta}\left(x_1, \cdots, x_T\right),$$

where P_d and P_{G_θ} are the real data and generated data distributions, respectively. The lower the value, the closer the model distribution is to the empirical data distribution.

Datasets

Two text datasets were used in our experiments for text generation are

- **The MS-COCO image captions dataset** (Chen et al., 2015) includes 4,682 unique words with the maximum sentence length 37. Both the training and test data contain 10,000 text sentences.
- **The EMNLP2017 WMT News dataset** (Guo et al., 2018) consists of 5,119 unique words with the maximum sentence length 49 after using first 10,000 sentences from (Nie et al., 2019). Both the training and test data contain 10,000 sentences.

Implementation and results

For MS-COCO dataset, we use policy gradient baseline value $b(s_t) = 2.5$ and $\alpha = 2.0$ for both Vanilla-RL and our model. The number of Mone Carlo samples we use during training is 3. For EMNLP News, we use $b(s_t) = 2$ and 5 Monte Carlo samples. In all experiments, we use gradient clipping value of 10.0 for the generator. In Tables 4.2 and 4.3 we report the means and standard deviations of test BLEU scores and training negative likelihoods values of our model compared to other baselines.

In Tables 4.2 and 4.3 we can see that, except for the OptiGAN-OnlyRL case, our model outperforms the baselines in BLEU scores on MS-COCO dataset and all but BLEU-2 for EMNLP News dataset. Our model also achieves a competitive NLL value with the best model, OptiGAN-OnlyGAN. This means that our model does not sacrifice the diversity of generated output when optimizing for the given score. We find that SeqGAN suffers the worst NLL score, even when compared to OptiGAN-OnlyRL. Since SeqGAN modified generator objective does not encourage matching the model distribution to data distribution, it can be susceptible to diversity loss. On the other hand, GANs that use Gumbel-Softmax to keep the standard generator objective, like ours, are more able to match the model to data distribution.

In the case of OptiGAN-OnlyRL, we find that pure reinforcement learning can achieve a higher BLEU score than other models (with very high variance). However, it has worse NLL values, which means it has worse diversity than our model. Fig. 4.3 shows that OptiGAN-OnlyRL fails to converge to low NLL, unlike our model, which has competitive NLL values with OptiGAN-OnlyGAN.

Qualitative Evaluation. To qualitatively evaluate the sentences from our model, we first show generated sentences, and compare the output with pure reinforcement learning baseline. In Table 4.4 we show generated sentences of OptiGAN, where we can see that the sentences generally look meaningful, structured and diverse, showing the capacity of OptiGAN in generating good and diverse sentences.

Although pure RL policy gradients baseline OptiGAN-OnlyRL can reach high BLEU scores, yet the sentences mostly are not realistic compared to our model. We show in Table 4.5 sentences from OptiGAN-OnlyRL, where we find that many of the generated sentences are unrealistic repetitions of certain n-grams in the test set. In the case of MS-COCO

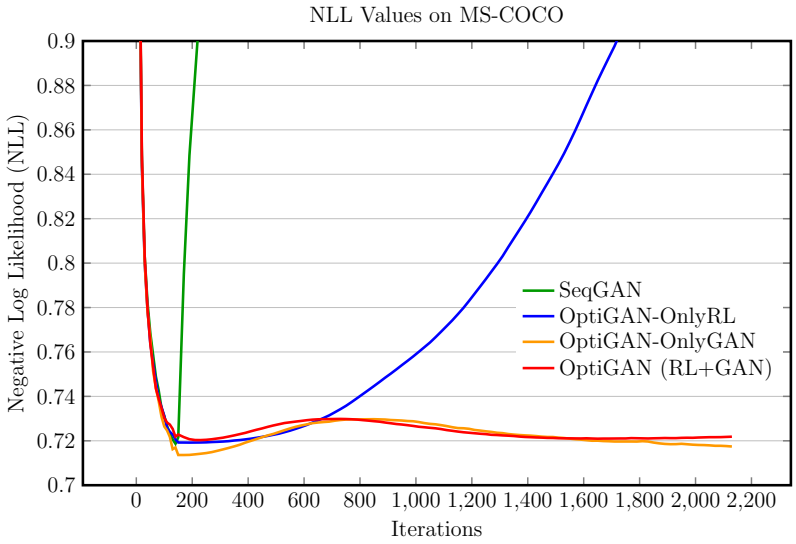

Figure 4.3: NLL values on MS-COCO Dataset. Unlike SeqGAN, OptiGAN does not sacrifice output diversity.

dataset, the generated sentences lengths are shorter than the average length of the dataset. This behavior possibly means that in the absence of the GAN objective part of the loss, pure reinforcement learning does not have incentive to generate sentences close to the real data distribution. In this case, the model only has to achieve high BLEU score to reduce the optimisation loss.

4.5.2 Air-Combat Trajectory Generation

Below we describe in details the experiments for OptiGAN with real-valued data on trajectory generation task.

Evaluation metrics

We use the McGrew score(McGrew et al., 2010) which measures how good is the aircraft positioned in an attempt to get behind the other aircraft. McGrew score is well-known by domain experts in air-combat maneuvers.

Dataset

For trajectory generation task, we used simulated data from *ACE-Zero* simulator (Ramirez et al., 2017). We created simulated trajectory data for the Stern Conversion maneuver (Austin et al., 1990) (Fig. 4.1) with two fighters; the blue and the red. We created 6,000

Table 4.4: Generated sample sentences from our model

Samples from MS-COCO
a roadside vendor sells food to passersby on there are two multitcolored towels .
an older man sitting at a kitchen with stainless steel appliances .
a woman standing in a field with mountains in the view of a field and a bus stop .
a clean bathroom with a blue toilet .
a group of people is watching buses next to a tall building .
a woman in a white shirt and jeans walking up a air gondola wears a costume decorations in a red jacket hides building
three small dogs under a towel rack .
a bathroom with a toilet and a large mirror
a city street with cars vehicles parked on the ground .
a large passenger jet flying through the air flying a kite and an airport .
Samples from EMNLP News
people had gone in a few weeks ago , it ' s really very quiet tonight to do .
she is me but it ' s a concept , the girls can build high strength .
i ' ve got a shock for their parents law to the same offence .
they ' re going to acknowledge that their football leader will be able to get every most republicans .
a tory source said : ' the 22 fall in the family in all of the newcastle day .
at cbs, that is more difficult, this is that the social stuff isn't quite any tribute for britain
it ' s a safe model from making a book of the first lady who had to respond .

Table 4.5: Sentences from OptiGAN-OnlyRL. Pure RL looses structure with BLEU rewards, repeating certain n-grams

the party has pledged a plate coach and don ' t think it was good to the real head .
and i ' ve been - it ' i ' i ' i have that thing a couple to hear the end but i ' m , " it ' s really people were going to do , ' it ' he work .
i ' d like , when i ' d like to give them to that it , there , and it ' ll happen looking to doubt that everybody challenges ...
if i ' ve got no evidence , it ' s it ' i expect ' there ' he ' he ' it ' he ' it ' out , and information it ' it ' ...
' i ' i ' it ' we ' i think i ' we spent a escape i ' i ' ve been on the ...

trajectories under this scenario [3].Each trajectory contains 16 featuresfor each of the two fighters.

Implementation and results

In all of our experiments, we use 40 simulation time steps (tokens) for each fighter trajectory. We use 256 units hidden layer for LSTM unit with 2 hidden layers. For the VAE part of the model, we use 12 hidden units and latent dimension of size 10. We pretrained the generator for 80 epochs before starting the adversarial and policy gradients training. In all experiments we set $\sigma = 0$ for sampling x_t. We can see from the generated trajectories is that the model is able to capture the correct behaviour, were the blue trajectory tries to get behind the red aircraft.

Table 4.6: Blue Fighter Engagement Scores (McGrew Score)

	λ	α	McGrew Score
SeqGAN*			N/A
LSTM	–	–	6.21
OptiGAN-OnlyGAN	1.0	–	7.34
OptiGAN	0.2	0.75	**8.41**
ACE0 Simulator Dataset	–	–	*8.53*

** SeqGAN works only with discrete data, not real-valued data*

Table 4.7: Effect of hyper-parameter λ for GAN-Only training

	λ	McGrew Score
OptiGAN-OnlyGAN	1.0	7.34
	0.2	6.79

Score Optimisation. We want the generated trajectories to be more optimized towards better engagement position against the red fighter. The desired outcome is a higher McGrew score, which means better engagement positions along the generated trajectory. We evaluate the effect of using policy gradients on the McGrew score of the blue aircraft and show the results in Table 4.6. In all experiments, we use $\gamma = 0.9$.

We compare with three baselines, Our model for real-valued data without adversarial training or policy gradients (LSTM), GAN without policy gradients (OptiGAN-OnlyGAN), and the average McGrew score of the training data (from the simulator). For all baselines,

[3]All trajectory data is available at `https://bit.ly/33k1AkT`

we generate 6,000 trajectories for evaluation. We can see that full OptiGAN model with the policy gradients achieves higher McGrew scores than other baselines, and closest to the real physics simulator. Although the GAN without policy gradients was able to achieve a slightly less score, the policy gradient model was run with the small value $\lambda = 0.2$. This means that the adversarial training did not contribute to the high score achieved by policy gradients model, rather, it was mainly the effect of policy gradients. As shown in Table 4.7, GAN with no PG model with $\lambda = 0.2$ did not achieve the same score as the one with $\lambda = 1.0$.

Qualitative Evaluation. To qualitatively evaluate the generated trajectories from our model, we show three different output categories form our model; *i) free, ii) conditional and iii) novel trajectories*. First, we train the model to generate *"free" trajectories*, in which the model freely generates both red and blue trajectories, without any input trajectory condition. In Fig. 4.4 we show free trajectories compared to real trajectories. It can be observed that the model can generate trajectories resembling the same behaviour in the real data.

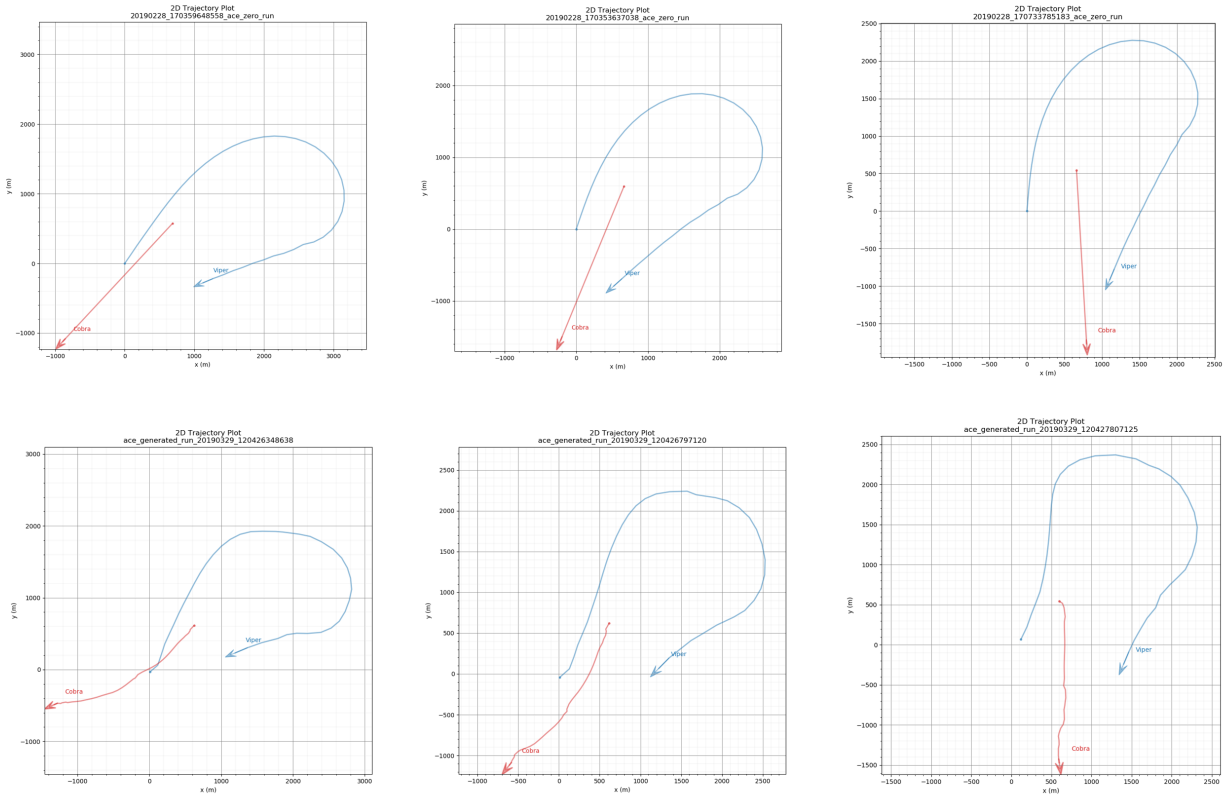

Figure 4.4: Samples of the training data and generated trajectories from the model. *Top row:* samples from the training trajectories in 2D position plane. *Bottom row:* generated trajectories from the trained model (McGrew score = 6.03).

Conditional Trajectories. Second, we develop a conditional variant of our model,

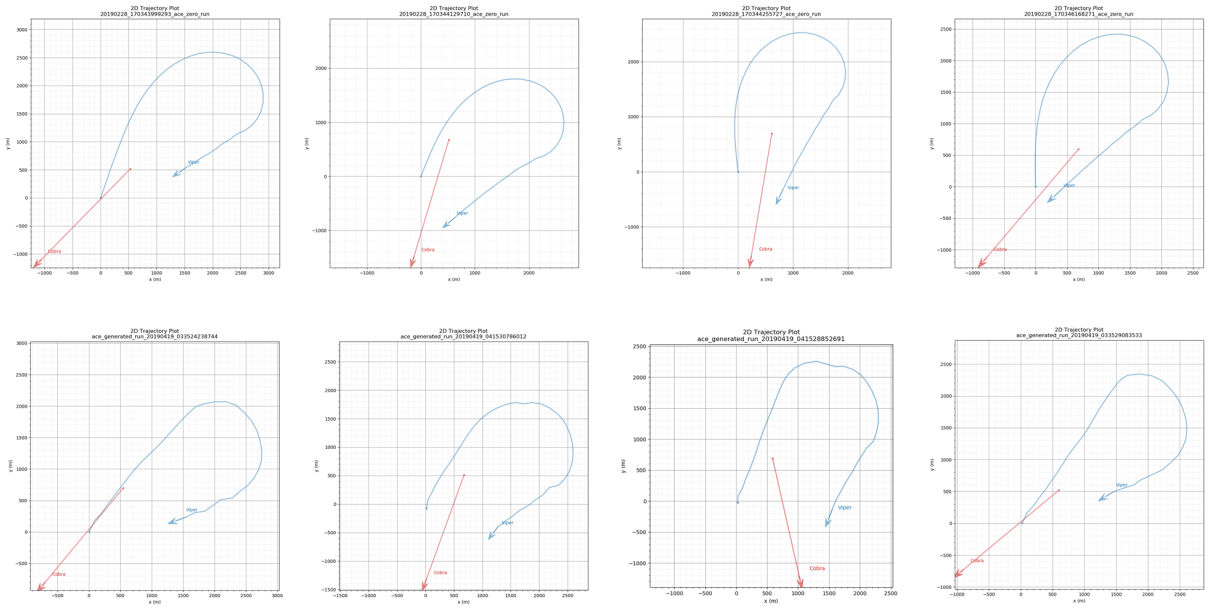

Figure 4.5: Samples of the training data and generated trajectories from the conditional model. *Top row:* samples from the training trajectories in 2D position plane. *Bottom row:* generated trajectories from the trained model.

where we want the model to be able to generate a trajectory for the blue aircraft given an input red trajectory. We show samples of the conditional trajectories in Figure 4.5, where we can see that the generated blue trajectory has a very close visual behaviour to the corresponding real blue training trajectory (for each column, the top is a training trajectory, and the conditional output is in the bottom). This shows that the model was able to correctly capture the true blue behaviour based on the true red input. In a real-world scenario, the conditional model can be used in real-time to suggest the best course of action for the blue aircraft given the current and past trajectory of the red aircraft.

Novel Trajectories. Finally, we want to try to discover novel blue aircraft behaviour, in the sense that the output trajectory resembles a behaviour that does not exist in training data. We show trajectories that exhibit novel behaviour generated by the model in Figure 4.6. These trajectories are not seen in the training data, in the sense that, the characteristics and shape of the trajectories are different. For example, in Figure 4.6, the scenario is set up in a way that the red flies straight to the South, and the blue flies North, but needs to turn around to tail the red. In the dataset, the shape of the blue trajectory in this scenario is consistent, where the first part of the flight (going North before the turn to the East) is always concave to the East. While in the generated trajectory, this first part is convex to

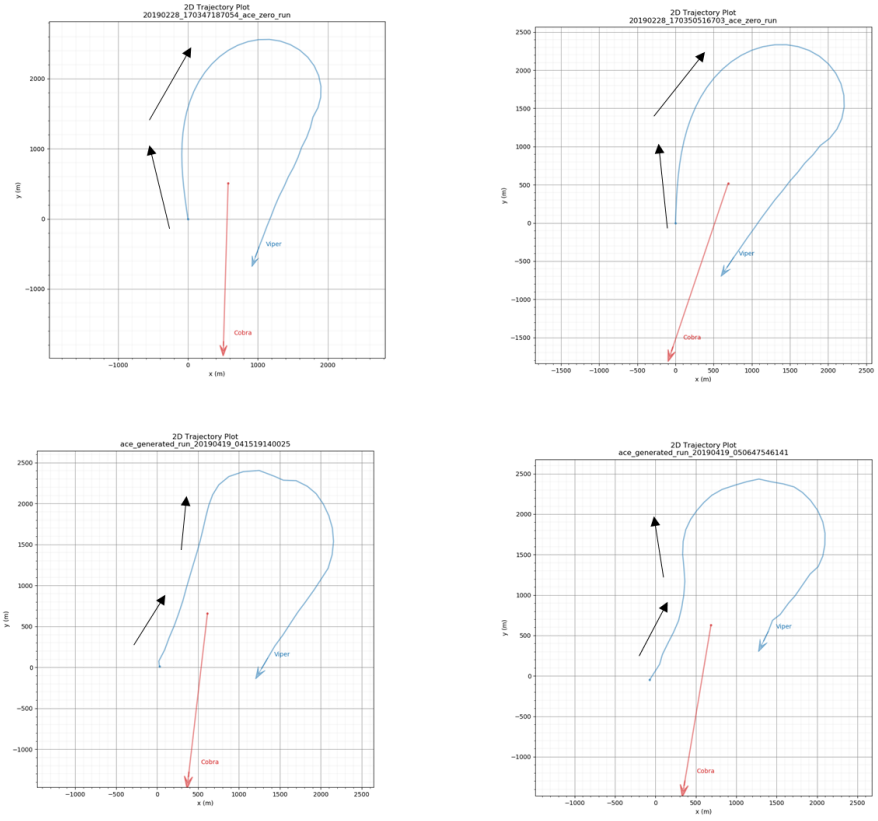

Figure 4.6: Examples of novel behaviour generated by the model different from training data. *Top row:* samples from the training data. *Bottom row:* Generated data from model

the East, and the convexity ends just before the turn. Such novel trajectories are potentially important for air combat scenarios, where usually fighters are trained for specific common scenarios, while in some situations, exploration of novel maneuvers might be important to achieve better combat effectiveness and increase the chance of successful outcomes.

4.6 Summary

In this chapter we addressed the problem of controlled generation in sequential generative models, in order to enable real-world use of generative models in different applications, including autonomous motion planning and drug or material design. In order to add this ability to sequence generative models, we extended our previous model (ARN) in Chapter 3 to include goal optimized generation mechanism in a new framework, named OptiGAN.

OptiGAN combines the diversity encouraged GAN framework in (ARN) with reinforcement learning (RL), since using RL methods like policy gradients allows maximizing accumulative future rewards. This way, OptiGAN can both learn the underlying data distribution through ARN objective, and gives the user a control mechanism that maximizes desired properties/goals. The final generator objective is combined MLE and GAN optimisation, and maximisation of the future rewards using REINFORCE policy gradients.

As a general framework, we conducted experiments on two tasks; text generation as discrete data, and air-craft trajectory generation as real-valued data. Our experiments show that OptiGAN can learn a distribution close the to the empirical (true) one, while achieving better desired scores compared the baselines. We conducted both quantitative and qualitative experiments to evaluate both discrete and real-valued tasks. We summarize the following findings:

- For text generation task, in the quantitative evaluation, the model achieved up to 3% higher absolute BLEU-4 scores over the baseline on MS-COCO dataset, and up to 3.11% on EMNLP dataset, as well as 20 relative improvement on BLEU-5, and with lower variance (Tables 4.2 and 4.3). OptiGAN also achieved equivalent or second best diversity over the baselines, where the diversity using NLL measure was improved by 12% and 6% over the RL baseline (SeqGAN) on MS-COCO and EMNLP datasets, respectively.

- Compared to relevant RL baselines like SeqGAN, OptiGAN outperformed in both desired scores and diversity, with much lower variance (around 50% decrease), requiring much less number of MC samples (reducing from 16 to less than 5 samples, an average of 75% decrease), and being faster to train.

- In the qualitative evaluation of text generation, we found that pure RL can sacrifice

sentence structure and coherence, tending to repeat certain tokens, in order to maximize the desired score. Thus, pure RL has no incentive to stay close to the empirical distribution if GAN term is not used in the loss, and tends to cheat the desired score by different means.

- For air-combat trajectory generation task, quantitative results showed that the model generated trajectories with higher engagement scores, compared to using only GAN objective without the RL objective (4.6). Qualitatively, we showed that a conditional variant of the model is able to generate proper maneuver trajectories based on a given input target trajectory. Finally, we showed the potential of discovering novel trajectory behaviours for real-world applications.

The comprehensive experiments we conducted showed that OptiGAN successfully addressed the lack of controlled or custom optimized generation in sequence generative models. Compared to RL baselines, the framework was significantly successful in learning the true data distribution, using ARN GAN-based objective. Our work in this chapter is one of the early models to address both the controlled generation and diversity of outputs in the sequential setting.

In order to further improve OptiGAN efficiency and simplicity, we look forward to overcoming the current limitations of our work. There are mainly two main limitations; first, the lack of leaned latent space for the discrete data. While the real-valued variant of OptiGAN in this chapter included a learned latent space, the specific implementation for discrete text data did not. A sufficiently disentangled latent space is an important feature for generative models that can be leveraged as a further fine control over outputs. Second, OptiGAN is currently sensitive to hyperparameters tuning, mainly in the policy gradients part, like the baseline computation for variance reduction.

In future work, we plan to address the limitations of the current OptiGAN framework. Specifically, we plan to *i)* make policy gradients baseline value learnable instead of using a fixed value, *ii)* improve the quality of real-valued outputs (e.g. trajectories) by adding realistic physical constraints from the domain, and extending to multi-agents setting, and *iii)* finally, we further look to incorporate disentangled latent space in the discrete case, which can be leveraged as a fine control over the generation process through disentangled factors of the data.

Chapter 5

Adversarial Attacks via Cross-Domain Interpretability

Training robust deep learning models for downstream tasks is a critical challenge. Research has shown that common down-stream models can be easily fooled with adversarial examples that look like the training data, but slightly perturbed, in a way imperceptible to humans. Understanding the behaviour of natural language models under these attacks is crucial to improve their robustness. For example, a prominent approach to defend deep learning models against malicious inputs is to find a set of adversarial examples that fool the classifier, and augment the training set with these examples to improve the model's generalisation. While early adversarial attack methods were "white-box", meaning they had access to model parameters, more interest started to grow for "black-box" attacks. In the black-box attack setting, the attacker has no access to the model parameters, and can only "query" the target model by providing the data to the model and then receiving its probability or label outputs to craft a successful attack. Nonetheless, current black-box state-of-the-art models are costly in both computational complexity and the number of queries needed to craft successful adversarial examples. For real world scenarios, the number of queries is critical, where less queries are desired to avoid detection or suspicion towards an attacking agent. In this chapter, we propose Explain2Attack, a query efficient black-box adversarial attack on text classification tasks. Instead of searching for important words to be perturbed by querying the target model, Explain2Attack employs an interpretable substitute model from a similar domain to learn word importance scores. Our framework either achieves or out-

performs attack rates of the state-of-the-art models, yet with lower queries cost and higher efficiency. [1]

5.1 Introduction

Achieving robustness for deep learning models against malicious inputs is a critical challenge. As previously discussed in Section (2.4), research has shown that common downstream deep learning models can be easily fooled with malicious inputs that look like the training data, but slightly perturbed, in a way imperceptible to humans. These perturbed inputs are called adversarial examples, which can be used to attack trained models, causing significant deterioration to down-stream task performance. There has been plenty of work on generating adversarial examples for different types of data, including images and text. The motivation behind is that the better we understand how a model is vulnerable to different attacks, the better we can increase its robustness. For instance, augmenting crafted adversarial examples in the training data can improve robustness of the models towards adversarial examples and improve its generalisation (Goodfellow et al., 2014b).

In general, attacks using adversarial examples can be crafted in either white-box or black-box settings. In white-box attacks, the attacker has access to the target model parameters, and the gradient of these parameters is used to craft adversarial examples (Belinkov and Glass, 2018; Wang et al., 2019; Yang et al., 2019; Sato et al., 2018). On the other hand, black-box attacks do not have access to the model parameters (Kuleshov et al., 2018; Gao et al., 2018; Jin et al., 2019), but only to its outputs. In this chapter, we are interested in black-box attacks, since, in practice, this is a more probable scenario for AI-systems deployed in the real-world.

Specifically, we consider in this chapter black-box attacks on natural language classification task. Typical classification models such as deep neural networks (DNN) output a probability distribution of their input belonging to each target class. Usually, the final label of the model is decided to be the one with the maximum probability. Hence, a classification model could be fooled if the confidence of the output probability is affected by a malicious input, switching the maximum probability to another incorrect target class.

[1] The work in this chapter was published at *the 25th International Conference on Pattern Recognition (ICPR 2020)*

We formulate the problem of crafting adversarial text attacks as a black-box conditional generation problem under language constraints, where preserving both the semantics (meaning) and syntax (structure) of a given input sentence after perturbation is challenging. The key strategy used to craft adversarial text in existing methods is to try to replace few words in an input sentence with synonyms such that its meaning remains the same. The classification model is then queried with these perturbed sentences to find out which ones successfully change the output label. Existing state-of-the-art models have different ways to search for most important words to replace, but the common intuition is to compute the importance score for each word as a function of the probability output of target model (see Section 5.3 for further details). Since existing approaches rely on word by word querying of the target model, they are costly in both computational complexity and number of queries. For real world scenarios, the number of queries is critical, where fewer queries are desired to avoid detection or suspicion towards an attacking agent.

In this chapter, we propose *Explain2Attack*[2], a black-box adversarial attack method on text classification, that employs cross-domain interpretability to learn word importance for crafting adversarial examples. The key idea is to replace the need to querying the target model by learning a similar substitute model with similar domain data, that can then be used to generate word importance scores for the targeted model. The advantages of our model are: *(i) less costly in computational complexity and number of queries, (ii) achieves or outperforms state-of-the-art methods in attack rates, yet with fewer number of queries, and (iii) scales significantly better in terms of queries needed for word importance computation with the length of input sequences.*

5.2 Background

Here we formally define the adversarial examples problem in the natural language domain, and the details regarding adversarial crafting process (Please refer to Section (2.4) for an overview on the general problem of adversarial examples). We discuss related work in details compared to our work in Section (5.3).

[2]Code is available at: `https://github.com/mahossam/Explain2Attack`

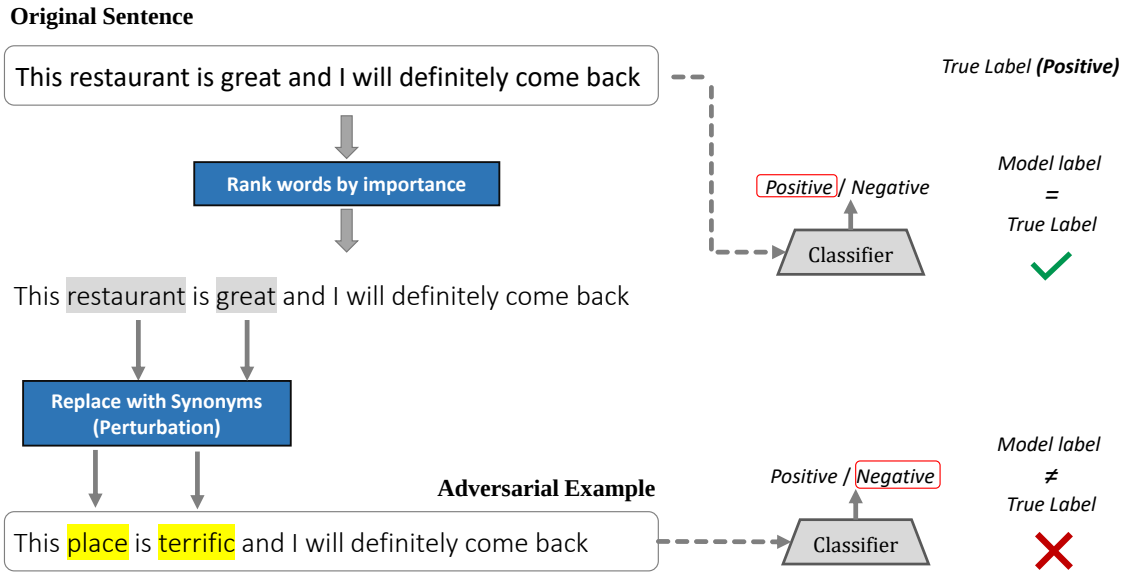

Figure 5.1: The standard procedure to craft a natural language adversarial example

5.2.1 Adversarial Examples in Natural Language

Let D be a dataset of N sentences and corresponding labels $D = \{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X} = \{X_1, X_2, ..., X_N\}$ is a corpus of N sentences, and $\mathcal{Y} = \{Y_1, Y_2, ..., Y_N\}$ is the collection of the class labels of M possible text classes. A pre-trained target model $F : \mathcal{X} \rightarrow \mathcal{Y}$ is the classifier model we want to attack. F maps the input space $\mathcal{X}$ to the label space $\mathcal{Y}$. Starting from an original sentence $X \in \mathcal{X}$, a valid adversarial example X_{adv} could be crafted such that:

$$F(X_{adv}) \neq F(X), \text{and } Sim(X_{adv}, X) \geq \epsilon, \tag{5.1}$$

where $Sim : \mathcal{X} \times \mathcal{X} \rightarrow (0, 1)$ is a similarity function and ϵ is the minimum desired similarity between the original and adversarial examples. In the case of natural language, this is usually a combination of semantic and syntactic similarity (Jin et al., 2019; Vijayaraghavan and Roy, 2019). An example of the detailed procedure to craft a natural language adversarial example is shown if Figure 5.1. Below we describe in details the main steps of the procedure.

Crafting Adversarial Examples

To craft an adversarial example for a given sentence $X \in \mathcal{X}$, the common strategy to follow is : i) selecting the most important words/tokens to replace from the input sentence, then ii) searching for synonyms to replace the most important words such that the changed sentence changes the classification label of the target model, iii) Finally, in order for the final adversarial example to be plausible and imperceptible to humans, the semantic similarity between the original candidate sentence and the final one need to very high (semantically close to each other) measured by some sentence similarity function $Sim(\cdot, \cdot)$.

Word Importance Ranking

Since the search space for all possible word placements to attack sentence X is large, most black-box attacks use a word importance ranking criteria, that helps prioritize which words in X to replace first. In details, let I_{w_i} be a score to measure the influence of a word $w_i \in X$ towards the model output probability $F_Y(X)$ of the predicted label Y. Different black-box methods vary on how to compute I_{w_i} for each word in a sentence, as discussed later in Section 5.3. However, they share the same need for the probability output of the classifier for class label Y, where I_{w_i} is computed as a function of these probability outputs:

$$I_{w_i} = \text{ScoreFunction}(F_Y(X)). \tag{5.2}$$

Word Replacement with Synonyms

After word ranking is done, word replacement step begins. In this step, the algorithm takes candidate words by order of importance, and replaces the current word by chosen synonyms till the target model label is changed. The candidate sentence X_{adv} with changed words is considered a valid adversarial example if it passes the condition in Eq.(5.1). After all word replacements are tried, if X_{adv} still could not change the label, then we assume that no adversarial example can be crafted from sentence X.

5.3 Related Work

There has been recent work on adversarial text attacks (Belinkov and Glass, 2018; Wang et al., 2019; Kuleshov et al., 2018; Yang et al., 2019; Sato et al., 2018; Gao et al., 2018; Ren et al., 2019; Jin et al., 2019) . The main challenges for natural language adversarial attacks are the discrete nature of inputs, where defining meaningful perturbations is not straight forward, and the search space and complexity for black-box attack methods.

Specifically for black-box text attacks, several methods (Jin et al., 2019; Ren et al., 2019; Gao et al., 2018) have been developed that share similar general framework, where the attack starts by selecting the most important words/tokens to replace from a candidate sentence, followed by searching for some word replacement that can flip the classification label of the target model. However, some methods followed the heuristic optimisation approach, for example, (Alzantot et al., 2018) used a genetic algorithm to find the best sentence perturbation that fools the classifier.

Most of the aforementioned black-box methods use the word selection/replacement strategy. For instance, PWWS (Ren et al., 2019) proposes computing a word saliency score using output probabilities of the target model, while (Gao et al., 2018) computes sequential importance score based on forward and backward RNN probabilities at the current word position in the sentence. TextFooler (Jin et al., 2019) is a recent strong baseline for text attacks, where the method uses a modified procedure for word ranking that increases the ranking in label disagreement case. BERT-Attack and BAE-Attack (Li et al., 2020; Garg and Ramakrishnan, 2020) both improve on TextFooler synonym replacement by using a pretrained language model to generate suitable substitute words based on the surrounding context. This achieved higher attack rates and the number of queries is reduced.

Our approach differs from previous work in solving the word ranking problem. Unlike other methods, instead of depending on the target model for word importance ranking, we *learn* word importance scores. The main differences of our approach compared existing ones are: i) word ranking has no dependence on the target model output, thus more efficient in the number of queries, ii) unlike existing methods, our approach is scalable for word importance computation with increasing sentence lengths, since computing the scores is not dependent on word by word query of the target model. This makes our approach more efficient, scalable, and less computationally expensive compared to existing methods. Moreover, our general

approach can benefit from further query reduction in the synonym replacement phase by incorporating the pretrained language model technique in BERT-Attack and BAE-Attack.

5.4 Proposed Framework

We propose a more efficient word ranking and selection model that alleviates the need for output probabilities for word ranking, hence, is more efficient in the number of needed queries of target model. Our approach is to build an interpretable substitute model that can closely resemble the target model behaviour on the attacked data domain. Then, using the interpretability capability of the substitute model, we can produce importance scores that can benefit our attack task.

In order to adapt our setup to work in a black-box setting, where we do not have access to attacked training data, we rely on the domain adaption capacity of deep models. Potentially, there is a similarity of language sentences representing a certain linguistic concept. For example, the words and semantic structures of the reviews for a restaurant and a movie can be usually similar, except that the subjects/objects of the reviews are different. Therefore, if a model is trained to capture such high-level features, the knowledge of the model can be transferred between datasets. This means that, if we do not have access to target model training dataset, we still can train a close enough *substitute* model on a *similar* dataset. In practice, deep learning models exhibit domain transfer capacity, where a model trained on certain data, can still behave well on other similar data that have similar high level features. Thus, attacks produced for the proposed substitute model (Papernot et al., 2017) can also be used to efficiently attack the original target model as long as they both are trained from similar domains. In Figure 5.2 we show the overview of our method.

In details, consider a target model F trained on some target training dataset $D_t^{\text{train}} = \{\mathcal{X}_t^{\text{train}}, \mathcal{Y}_t^{\text{train}}\}$ and testing set $D_t^{\text{test}} = \{\mathcal{X}_t^{\text{test}}, \mathcal{Y}_t^{\text{test}}\}$. Instead of querying target model $F(\cdot)$ with context around each word in sentences from $\mathcal{X}_t^{\text{test}}$, we consider learning a substitute interpretable model that can provide importance scores for given words. For this purpose, we leverage a framework extended from (Chen et al., 2018a) to train the substitute model.

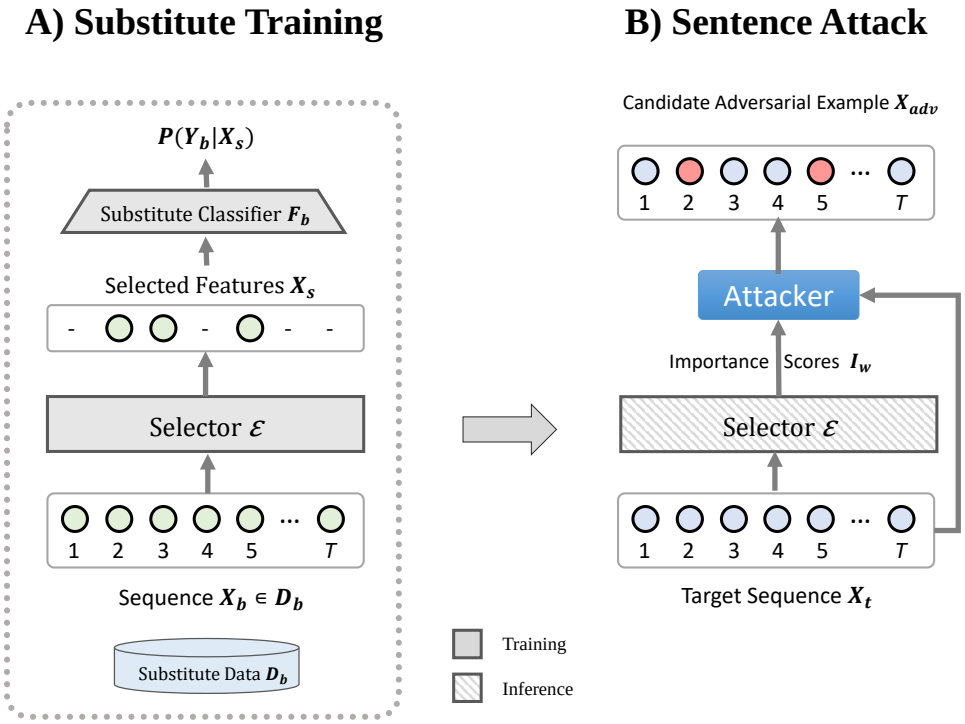

Figure 5.2: Overview of Explain2Attack Framework.

Interpretable Substitute Model

In the black-box attacks setup, we do not have access to target training data D_t^{train}. Therefore, in order to train a substitute model that can generate importance scores during attack, we need to look for another dataset that is close enough to the target model dataset. We call such a dataset the substitute dataset, $D_b = \{\mathcal{X}_b, \mathcal{Y}_b\}$. We then use D_b to train the substitute model we call SUB.

Our goal from training SUB is to learn a network (called the *selector*) that can select the most important features from the input $X \in \mathcal{X}_b$ to let another network, the *substitute classifier*, correctly predict the corresponding label $Y \in \mathcal{Y}_b$. In details, inspired by (Chen et al., 2018a), for a substitute pair $(X, Y) \sim D_b$, our goal is to learn a selector network $\mathcal{E}(X)$ that selects the most important subset of k features from X that is sufficient for substitute classifier $F_b(.)$ to correctly predict Y. After substitute training is finished, $F_b(.)$ can be discarded, since we are only interested in the selector $\mathcal{E}(X)$.

Formally, for a given positive integer k, let ρ_k be the set of all possible subsets of size k:

$$\rho_k = \left\{ S \subset 2^d \ : \ |S| = k \right\}.$$

We denote the selected k features indices as S, which and are drawn from the probability distribution P, the output of the selector $\mathcal{E}(X)$:

$$S \sim P(S|x) = \mathcal{E}(X),$$

where $S \in \rho_k$. Lastly, we denote the corresponding selected features sub-vector from X as $X_S \in \mathbb{R}^k$. The choice of the number of explaining features k can be tuned as a hyper-parameter. The learning objective is to find $\mathcal{E}$ that maximizes the mutual information $I(X_S; Y)$:

$$\max_{\mathcal{E}} I(X_S; Y) \quad \text{subject to} \quad S \sim \mathcal{E}(X). \tag{5.3}$$

Maximizing a variational lower bound

Since a direct solution to the Problem 5.3 is not tractable, an approximate solution can be found using a lower bound on the mutual information (Section 2.1.2). Starting from 5.3, $I(X_S; Y)$ can be expressed as:

$$\begin{aligned} I(X_S, Y) &= \mathbb{E}\left[\log \frac{P_b(X_S, Y)}{P(X_S)\,\mathrm{P}_b(Y)}\right] = \mathbb{E}\left[\log \frac{P_b(Y \mid X_S)}{P(Y)}\right] \\ &= \mathbb{E}\left[\log P_b(Y \mid X_S)\right] + \text{ Const.} \\ &= \mathbb{E}_X \mathbb{E}_{S|X} \mathbb{E}_{Y|X_S}\left[\log P_b(Y \mid X_S)\right] + \text{ Const,} \end{aligned}$$

where $P_b(Y)$ is the empirical distribution of the labels from D_b. Since it is not possible to compute expectations under $P_b(Y \mid X_S)$, a variational approximate distribution $Q_S(Y \mid X_S)$ can be used for the optimisation, where:

$$\mathbb{E}_{Y|X_S}\left[\log P_b(Y \mid X_S)\right] \geq \mathbb{E}_{Y|X_S}\left[\log Q_S(Y \mid X_S)\right].$$

This way, the maximisation in 5.3 can be carried out on the variational approximate:

$$\max_{\mathcal{E},Q} \mathbb{E}\left[\log Q_S(Y \mid X_S)\right] \quad \text{such that } S \sim \mathcal{E}(X). \tag{5.4}$$

In the attack step, in order to generate word importance scores from the trained SUB, we take the logit output of the last layer of $\mathcal{E}(X)$ before the k selection process $S \sim \mathcal{E}(X)$. This way we obtain importance score vector $I \in \mathbb{R}^d$ that has the same dimensions of the

input.

The selector $\mathcal{E}$ takes the form of a multilayer deep convolutional neural network. In order for $\mathcal{E}$ to select k features from X, a Gumbel-Softmax layer (Jang et al., 2016a) is used to generate k indices (please refer to the basic formulation described in Section 3.3.3).

Adversarial Examples via Cross-Domain Interpretability

At inference time, when crafting the adversarial attacks, we do inference on SUB using the target testing set D_t^{test}. This is the standard setup for black-box attack, where D_t^{test} is used as starting point and slightly perturbed to craft valid adversarial examples. As described earlier, because of the domain adaption capacity of deep learning models, the closer the substitute domain is to the target, the better the generated scores will be for the original target model.

Implementation

Our attack method is inspired by the framework proposed in (Jin et al., 2019). However, we modified the behaviour so that we alleviate the need for querying $F(\cdot)$ for word ranking, since our proposed method relies on interpretability architecture for this purpose. We name our framework *Explain2Attack*. In Algorithms 5 and 6 we describe in details how our algorithm works.

Algorithm 5: Train Substitute Model SUB

Input: Substitute training corpus $\mathcal{X}_b = \{X_1, X_2, ..., X_N\}$ of N sentences, and corresponding class labels $\mathcal{Y}_b = \{Y_1, Y_2, ..., Y_N\}$

Output: Trained substitute model SUB with selector network $\mathcal{E}$

for *each mini batch* $B_s \in \{\mathcal{X}_b, \mathcal{Y}_b\}$ **do**
  Update parameters of SUB model to find $\mathcal{E}$ that maximizes objective in Eq. (5.4)
end

We follow Algorithm 5 to train the substitute model SUB using D_b. After that, selector $\mathcal{E}$ from SUB can be used in algorithm 6 for crafting adversarial examples.

The procedure to craft adversarial examples is described in Algorithm 6. The algorithm starts from an input sentence X_t^{test} and terminates either after successfully finding a perturbed adversarial example X_{adv} that changes the label Y_t^{test}, or if no such perturbation is

Algorithm 6: Explain2Attack: Crafting Text Adversarial Examples

Function *Explain2Attack*:

Input: Target test sentence example $X_t^{\text{test}} \in \mathcal{X}_t^{\text{test}}$, where $X_t^{\text{test}} = \{w_1, w_2, \ldots, w_n\}$, the corresponding ground truth label Y_t^{test}, target model F, sentence similarity function $Sim(\cdot, \cdot)$, sentence similarity threshold ϵ, word embeddings Emb over the vocabulary **Vocab**

Output: Perturbed Adversarial example X_{adv}

Initialization: $X_{adv} \leftarrow X_t^{\text{test}}$

for *each word index* $\boldsymbol{i}$ *in* X_t^{test} **do**

  Retrieve word importance score $I_{w_i} = \textbf{Get\_Word\_Score}(X_t^{\text{test}}, \boldsymbol{i})$

end

$W \leftarrow$ Sorted list of words in $w_i \in X$ in descending order of importance score I_{w_i}

for *each word* $w_j \in W$ **do**

  Candidates $\leftarrow \{\,\}$

  Extract top M synonyms of w_j from **Vocab** by highest cosine similarity:

  $$\text{Synms}(w_j) \leftarrow \Big\{ \text{v} \in \textbf{Vocab} : \Big| \big\{ \text{v}' \in \textbf{Vocab} : \text{CosSim}\left(Emb_{w_j}, Emb_{\text{v}}\right) > \text{CosSim}\left(Emb_{w_j}, Emb_{\text{v}'}\right) \big\} \Big| < M \Big\}$$

  for *each synonym* $c_m \in Synms(w_j)$ **do**

    $X'_m \leftarrow$ Replace w_j with c_m in X_{adv}

    Add X'_m to **Candidates**

  end

  if $\exists\, X' \in$ **Candidates** *s.t.* $F(X') \neq Y_t^{test}$ *and* $Sim\left(X_t^{test}, X'\right) \geq \epsilon$ **then**

    $$X_{adv} \leftarrow \underset{X' \in \textbf{Candidates}}{\text{argmax}} \; Sim\left(X_t^{\text{test}}, X'\right)$$

    return X_{adv}

  else if $F_Y\left(X_{adv}\right) > \min\limits_{X' \in \textbf{Candidates}} F_Y(X')$ **then**

    $$X_{adv} \leftarrow \underset{X' \in \textbf{Candidates}}{\text{argmin}} \; F_Y(X')$$

  end

end

return None

Function *Get_Word_Score*(X_t, i):

  word_score $= \mathcal{E}_{\text{logits}}(X_t)_i$

  return word_score

found. In details, the algorithm proceeds by inferring from $\mathcal{E}$ word scores for every word in X_t^{test}. These scores are sorted, called W, and then the algorithm tries to replace word by word

to find valid X_{adv}. For each word w_j in W, a set of top M synonyms are extracted from the vocabulary (**Vocab**) based on the embedding cosine similarity CosSim($Emb_{w_j}, Emb_{\text{word}\in\text{Vocab}}$) as in (Jin et al., 2019). Then a set of the candidate adversarial sentences (**Candidates**) is created by replacing w_j with each extracted synonym.

At the current word w_j, the algorithm first checks if perturbing w_j in X_{adv} using the set of synonyms in **Candidates** can change Y_t^{test}. If one or more such synonym are found, the one which achieves the highest similarity $Sim\,(X_t^{\text{test}}, X')$ to original sentence is picked, and the algorithm terminates. *Otherwise*, if no synonym is found that can change Y_t^{test}, the algorithm needs to choose the synonym perturbation that yields the weakest (minimum) target model probability $F_Y(X')$ before moving on to the next word w_{j+1}.

Semantic Similarity Check $\boldsymbol{Sim\left(X_t^{\text{test}}, X'\right)}$. Following (Jin et al., 2019), we use the Universal Sentence Encoder (USE) to encode the original and the adversarial candidate sentences into a high dimensional vector space, and use their cosine similarity score the semantic similarity between X_t^{test} and X'.

5.5 Experiments

We report here the results of our method on text classification tasks. We apply our framework to several sentiment classification datasets with WordCNN, WordLSTM and BERT (Devlin et al., 2018) target models. However, our model can be applied to other classification models or datasets with a proper choice of substitute datasets . We compare our results to TextFooler (Jin et al., 2019), a state-of-the-art baseline for black-box text attack on the chosen target models. Below we describe the datasets, metrics and discuss the results. In Table 5.1 we report the datasets we used in our experiment with their statistics.

Datasets

- **IMDB** and **MR**: Movie reviews for sentiment classification (Maas et al., 2011a; Pang and Lee, 2005). The reviews have binary labels, either positive or negative.
- **Amazon MR**: Amazon polarity (binary) user reviews on movies, extracted from the larger Amazon reviews polarity dataset [3].

[3] `https://www.kaggle.com/bittlingmayer/amazonreviews`

- **Yelp Polarity Reviews**: Sentiment classification on positive and negative businesses reviews (Zhang et al., 2015). We mainly use this dataset as a substitute dataset when attacking other models.

In all of the datasets except Amazon MR, we follow the data preprocessing and partitioning in (Jin et al., 2019).

Table 5.1: Statistic of Used Datasets

Dataset	Train	Test	Avg. Length
IMDB	25K	25K	215
MR	9K	1K	20
Amazon MR	25K	25K	100
Yelp	560K	38K	152

Metrics

We evaluate our method using the following metrics:

- **After-attack accuracy** (Adv_Acc): We report for each model the original clean accuracy Clean_Acc of the target model on the test set. Then we report the accuracy of the target model against crafted adversarial examples Adv_Acc. The lower is this better, where the larger the gap between these two accuracies means more successful the attack method. Through-out discussion, when we refer to "***attack rate***", we mean the gap (Clean_Acc − Adv_Acc).

- **Average number of queries** (Avg_Queries)**:** We report the average number of queries needed to find successful adversarial example per input sentence. The lower is better, where fewer number of queries is one of the main desired goals of our method. This is an absolute measure of the number of queries regardless of the achieved after-attack accuracy.

- **Query Efficiency** (QE)**:** Since more successful attacks need more queries in black-box setting, we cannot rely only on Avg_Queries for evaluating the performance of our method. We need to measure the true benefit in reduction of number of queries, and make sure that it is not reduced because of sacrificing attack rate. This is the

Table 5.2: After-Attack Accuracies, Queries and Query Efficiency

Classifier		BERT		WordCNN			WordLSTM		
Target Model		IMDB	MR	IMDB	MR	Amazon MR	IMDB	MR	Amazon MR
	Clean_Acc.	92.18	89.97	87.32	79.85	90.14	88.78	81.82	91.30
Adv_Acc. ↓	TextFooler (Jin et al., 2019)	11.88	13.59	**0.60**	1.50	**3.92**	**0.04**	**2.06**	**2.15**
	(Substitute Data)	*(Yelp)*	*(Amazon MR)*	*(Yelp)*	*(IMDB)*		*(Amazon MR)*		*(IMDB)*
	Explain2Attack (ours)	**11.32**	**13.34**	0.61	**1.31**	3.97	0.06	2.27	2.38
Avg_Queries ↓	TextFooler	980.5	**181.6**	444	112.8	378.7	500.2	117.5	392.7
	Explain2Attack	**873.5**	184.07	**404.5**	**108.7**	**349.4**	**440.5**	**114.2**	**369.3**
Query Efficiency (QE) ↑	TextFooler	0.082	**0.421**	0.195	0.695	0.228	0.177	0.679	0.227
	Explain2Attack	**0.093**	0.416	**0.214**	**0.723**	**0.247**	**0.201**	**0.697**	**0.241**

motivation behind Query Efficiency (QE), the ratio of successful attacks per query $QE = \frac{\text{Clean\_Acc} - \text{Adv\_Acc}}{\text{Avg\_Queries}}$. The higher is better. This means that QE measures the percentage of successful attacks per single query, hence the true query efficiency related to the attack rate.

- **Perturbation Query Cost (PQC)**: The number of queries needed per perturbed word $PQC = \frac{\text{Avg\_Queries}}{\text{Pertureb\_Words}}$. The lower is better. PQC measures the cost in terms of queries needed for a useful word perturbation that contributes towards label change.

Results and Discussion

The main focus in evaluating the performance of our method is the number of queries, both as absolute measure and its efficiency relative to the achieved attack rate. These two aspects of evaluation are critical to measuring the true gains we get from our approach.

First, in Table 5.2, we compare after-attack accuracies and corresponding average number of queries to the baseline. For each target model and dataset, we report the best substitute dataset that yielded the best result. Across all but one case, we find that our method either achieves or out-performs attack rates of the baseline, yet with lower number of queries. The gains in terms of the number of queries differ according to the different datasets. We discuss this in details later in this section.

Lastly, it is important to study the cases where our method achieves a lower number of queries but does not achieve higher attack rate than the baseline. In the black-box setting, higher attack rate is related to the number of queries, since there is a cost of a certain number of queries for every word perturbation. This means that lower number of queries is only beneficial if it improves or preserves the attack rate. Therefore, in order to measure the true efficiency of our method in terms of queries per successful attack, we report in Table

Table 5.3: Effect of Sentence Length on Number of Queries

Target Dataset		IMDB			Amazon MR		MR		
Classifier		BERT	CNN	LSTM	CNN	LSTM	BERT	CNN	LSTM
Average Sentence Length		215			100		20		
Avg_Queries ↓	TextFooler	980.5	444	500.2	378.7	392.7	112.8	117.5	**181.6**
	Explain2Attack	**873.5**	**404.5**	**440.5**	**349.4**	**369.3**	**108.7**	**114.2**	184.07
Difference		106.5	39.5	59.7	29.3	23.4	4.1	3.3	-3.0

5.2 Query Efficiency ratio QE. We find that when our method has fewer number of queries, it has better QE ratio, even if the attack rate is not better than the baseline. This implies that the lower number of queries achieved comes from true efficiency of our method, not because of the corresponding drop in attack rate.

Effect of Sentence Length In the case of MR dataset in Table 5.2, we find that the reduction in the number of queries is the lowest among other datasets. By comparing the statistics of the datasets (Table 5.1) with the results in Table 5.2, we find a correlation between the dataset average sentence length and the corresponding reduction in attack queries. The longer the sentences are, the more reduction our method achieves in number of queries. We summarize this finding in Table 5.3. This effect is in agreement with our design goals, since unlike other existing methods, our method does not perform word by word ranking through target model querying. Hence, with longer input sentences, the reduction in the number of queries is improved by our method. This is a key advantage of our method in terms of scalability to datasets with longer sentences.

Query Cost of Perturbed Words Another important measure is the number of queries needed for every word perturbation; Perturbation Query Cost (PQC). We are interested in this measure in order to understand the added cost of queries if the model requires more words to be perturbed to find a successful attack. In Table 5.4 we report PQC against the baseline. In addition, we report both the maximum and average number of perturbed words needed for successful adversarial attack. Results show that our method outperforms the baseline in PQC measure, meaning that our method scales better with the number of perturbations needed for successful attacks.

Table 5.4: Perturbation Query Cost **(PQC)**

Classifier		BERT		WordCNN			WordLSTM		
Target Model		IMDB	MR	IMDB	MR	Amazon MR	IMDB	MR	Amazon MR
Max Perturbed Words	TextFooler	222	20	56	14	**35**	89	**13**	75
	(Substitute Data)	*(Yelp)*	*(Amazon MR)*	*(Yelp)*	*(IMDB)*		*(Amazon MR)*		*(IMDB)*
	Explain2Attack	**127**	20	**54.3**	**11**	41.7	**84.7**	13.8	**68**
Avg. Perturbed Words	TextFooler	18.7	4.2	5.8	2.3	5.0	7.6	2.5	7.0
	Explain2Attack	22	4.8	9.1	2.8	6.5	11.3	2.9	8.7
Perturbation Query Cost (PQC) ↓	TextFooler	52.5	43.5	76.6	49.0	76.3	65.8	47.0	56.4
	Explain2Attack	**39.7**	**38.6**	**44.5**	**38.8**	**53.8**	**39.2**	**39.2**	**42.2**

Table 5.5: Adversarial Examples Sentences. Perturbed Words are Highlighted

	Label	Sentence
Original	(0=Negative)	the film lapses too often into sugary **sentiment** and withholds delivery on the pell mell pyrotechnics its punchy style promises
Adversarial	(1=Positive)	the film lapses too often into sugary **emotions** and withholds delivery on the pell mell pyrotechnics its punchy style promises
Original	(1=Positive)	the movie sticks much closer to hornby 's drop dead confessional tone than the film version of high **fidelity** did
Adversarial	(0=Negative)	the movie sticks much closer to hornby 's drop dead confessional tone than the film version of high **faithful** did
Original	(0=Negative)	i'm not exactly sure what this **movie thinks** it is about
Adversarial	(1=Positive)	i'm not exactly sure what this **cinematography concepts** it is about
Original	(1=Positive)	the performances of the four **main** actresses bring their characters to life a **little** melodramatic , but with enough **hope** to keep you engaged
Adversarial	(0=Negative)	the performances of the four **underlying** actresses bring their characters to life a **littlest** melodramatic , but with enough **wanting** to keep you engaged

Qualitative Assessment We show in Table 5.5 examples of successful adversarial examples from our method. We find that the language semantic is preserved, and that the choice of perturbed words resembles important keywords that contribute to the original label.

Transferability Conditions In all of our experiments, we used the standard L2X (Chen et al., 2018a) architecture as the choice for the substitute architecture. However, we found that changing the substitute architecture affects attack rates. This finding is similar to (Demontis et al., 2019; Madry et al., 2018) where attack performance is found to be related to the model's architecture and complexity. We further look to study in details why and when attacks transfer well between substitute and target models. However, we leave such comprehensive study for future work.

5.6 Summary

In this chapter, we proposed a general framework that employs interpretability across domains for crafting black-box text adversarial attacks. We consider the problem of conditional generation of adversarial sentences from a given input sentence, where the goal is to perturb the input while preserving its semantics and structure. In a typical the black-box setting, the main challenge we address is the query and computation cost of adversarial sentences. The main intuition is to learn word ranking that most probably impacts the target model instead of searching for it expensively.

Most of the existing methods depend on heavily querying the target model to choose which words to perturb with replacement synonyms. We proposed a novel approach to address this problem through domain transfer capacity, named *Explain2Attack*, in order to reduce the number of queries needed, and increase the computational efficiency. Specifically, we train an interpretable substitute model on a substitute dataset, with no need for the target dataset. The substitute model learns word importance scores for the word ranking on the substitute domain. After the substitute training, the substitute model is then used in the word ranking step in the target domain to generate word importance scores, with fewer number of queries and higher efficiency.

To evaluate the performance of *Explain2Attack*, we utilize common metrics including number of queries and successful attack rates. In addition, we propose two new metrics: Query Efficiency ratio (QE) and Perturbation Query Cost (PQC). Both measure the true effectiveness of attack methods in terms of number of queries per successful attack or per word perturbation.

- Our method either achieves or out-performs attack rates of the baseline, yet with lower number of queries. [In Table 5.2, on IMDB dataset, 11.9%, 11.7% and 8.7% reductions in number of queries are achieved compared to the baseline (measured on attacks per query unit (QE)), with BERT, LSTM, and CNN models respectively.]

- With lower number of queries, our method has better Query Efficiency ratio (QE), even when the attack rate is lower than the baseline. Hence, fewer queries achieved come from inherent query efficiency of our method.

- A key advantage of our method is that it is more scalable for word importance com-

putation with the input length, since it does not perform word by word ranking. The longer the sentences are, the more reduction our method achieves in needed queries. Table 5.3 shows 13%, 7% and 6% improvements in queries reduction when input length increased from 20 to 215 words, on LSTM, BERT, and CNN models, respectively.

In addition, to measure the queries efficiency relative to the number of perturbed words, we propose Perturbation Query Cost (*PQC*). *PQC* is the number of queries needed for every word perturbation. In Table 5.4, on IMDB dataset, our method needs 26, 32 and 12 fewer queries per word perturbation than the baseline. Results show that our method scales better with the number of perturbations needed for successful attacks.

Through extensive experiments, we show that our proposed framework either achieves or out-performs attack rates of the state-of-the-art baseline, yet with lower queries cost and higher efficiency. However, our work has some limitations. First, the framework needs probability outputs from the target model. Although most current natural language attack methods have the same limitation, other attack methods on other types of data like images have alleviated this limitation (Papernot et al., 2017). Second, the synonym ranking and replacement procedure requires a significant number of queries to the target model. We find that this part is responsible for the most queries needed for a successful attack. Therefore, changing this procedure is very important for any further improvement to our framework. Finally, the choice of the substitute domain data is mainly dependent on human experience and experiments. For a real-world usage of our framework, simplifying or automating the choice of the substitute domain is useful.

For future work, we plan to overcome the limitations of our current framework. Specifically, we plan to *i)* train the selector network to directly attack the target classifier through a reward signal feedback, in order to improve attack rate, *ii)* study and formalize transferability conditions from substitute to target domains, and provide guides for the suitable choice of substitute models and domains, and *iii)* further reduce dependence on target model by choosing replacement synonyms through a pretrained language model (Li et al., 2020). Therefore, we can further reduce the need for target model probabilities.

Chapter 6

Learning Adversarial Attacks with Target Model Guidance

In the previous chapter, we introduced a learning approach for adversarial attacks on natural language classification models. The key idea was to replace the query-expensive search for important words ranking with a substitute model that can learn the word ranking. We thus showed that effective word ranking is possible without the need to extensively query the target model during each attack. In that black-box setting, it was desired to limit the access to the target model and data as much as possible, and thus the learning did not benefit from the target model's information, since no access to the target data was possible and with limited access to its outputs. Therefore, the learning could only be performed on another dataset, which we call the substitute domain dataset. As shown in the previous chapter, this framework achieved comparable attack rates to the baseline with fewer queries. Nonetheless, it is generally desirable to perform the word ranking learning directly on the target data domain, and to benefit from the target model's output information as a learning signal for the substitute model, in order to generate better adversarial examples. In this chapter, we investigate different methods to incorporate such learning signal from the target classifier. Through a preliminary set of experiments, we show that the proposed methods can improve both attack rates and average number of queries. Finally, we provide potential future directions for further improvements.

6.1 Introduction

Most of the existing black-box attack methods on natural language models directly query the target model to rank sentence words for replacement. This procedure is expensive in terms of number of queries needed to the target model, and the number of queries increases with the increased input length. In addition, the increased number of queries to the target classifier is not desirable in black-box settings, where it can raise the suspicion towards the attacking agent. To address this issue, we presented *Explain2Attack* in Chapter 5, where an interpretable model is trained to learn good word ranking for synonym replacement.

In Explain2Attack model, two main network components are trained; the *substitute classifier* F_b, and the *selector* network $\mathcal{E}$. The substitute classifier F_b is trained as a substitute for the target classifier with the cross-entropy loss on the substitute data and labels $D_b = \{\mathcal{X}_b, \mathcal{Y}_b\}$, while the selector $\mathcal{E}$ is the main component we are interested in to select the important words, and its parameters are updated to minimize the same loss for F_b . Under this architecture, Explain2Attack was able to achieve comparable attack rates to the baseline while reducing the computational and query complexity.

However, the word ranking step in Explain2Attack was trained on the substitute dataset and classifier, which are both different from the target classifier and the target dataset. The substitute training was designed in this way in order to limit the access to the target model's through extensive queries, and to the training dataset D_t^{train} . The key intuition is that as long as the chosen dataset for the substitute training dataset D_b comes from a similar domain to the target training dataset, then the learned word ranking model should perform good enough on test sentences from the target dataset.

Although this setting leads to competitive attack rates and reduced queries if satisfied, it is still a restriction on the substitute training procedure. Additionally, in some scenarios, the access to target model's data might be possible or available. Thus, by relaxing restrictions on the access to possible target model information (both outputs and data domain), we might be able to achieve better attack rates with a marginal overhead of additional queries used for the substitute training. It is therefore desirable to perform the substitute learning directly using the target model outputs or data domain, in order to leverage all possible information to generate better adversarial examples and achieve better queries efficiency in terms of the average number of queries per attack.

In this chapter, we propose two modifications to Explain2Attack to better train the selector network while benefiting from the target classifier output information according to the availability of the target test set. Specifically, we do not use the labels used for selector training in Chapter 5, which came from the substitute dataset $\mathcal{Y}_b$. Instead, we replace them with output predictions from the target classifier under two settings: *i)* the target model is given the substitute dataset sentences $\mathcal{X}_b$, or *ii)* the target model is given the target test set $\mathcal{X}_t^{\text{test}}$. In the following sections, we describe in details the two modifications and their intuitions, and provide experimental results for different natural language classification target models and datasets. Finally we provide discussion and insights for this research problem and potential future directions.

6.2 Method

In order to employ the target model predictions in the substitute model training, we consider two black–box settings: first, substitute training without access to the target test set, and second, substitute training with access to the target test set.

In details, we first revise the original setting of Explain2Attack from Chapter 5: let a target model F_{target} be trained on some target training dataset $D_t^{\text{train}} = \left\{\mathcal{X}_t^{\text{train}}, \mathcal{Y}_t^{\text{train}}\right\}$ and testing set $D_t^{\text{test}} = \{\mathcal{X}_t^{\text{test}}, \mathcal{Y}_t^{\text{test}}\}$. We then train a substitute model, SUB, to learn the word importance scores. SUB is trained using a dataset that is close enough to the target model dataset called the substitute dataset, $D_b = \{\mathcal{X}_b, \mathcal{Y}_b\}$. The substitute model itself contains two sub-networks called the *substitute classifier* F_b and the *selector* network $\mathcal{E}_\theta$. After training, performing inference on the selector network $\mathcal{E}_\theta(X)$ using input sentences yields the desired word importance scores for these sentences. During the substitute training procedure, the substitute classifier F_b is trained to correctly predict the label $Y \in \mathcal{Y}_b$ from an input sentence $X_s \sim \mathcal{E}_\theta(X)$, where $X \in \mathcal{X}_b$, and X_s is the most important selected k words for some input X. After substitute training is finished, $F_b(.)$ can be discarded, since we are only interested in the selector $\mathcal{E}(X)$.

To this end, we change the original setting of Explain2Attack to incorporate the target classifier predictions outputs in the selector network training. Specifically, we want to replace the labels $\mathcal{Y}_b$ that come from the substitute dataset D_b with output labels or probabilities from the target classifier F_{target}. This way, the training of F_b, and most importantly, the

selector $\mathcal{E}_\theta$, will be trained such that it learns or imitates the behaviour of the target classifier. This requires querying the target model $F_{\text{target}}\left(\widetilde{X}\right)$ with some input $\widetilde{X}$ to return target classifier predictions. The choice of the inputs to the target model $\widetilde{X}$ allows to different settings: *i)* We either choose $\widetilde{X}$ to come from the substitute dataset $\mathcal{X}_b$, or *ii)* from the target test-set sentences $\mathcal{X}_t^{\text{test}}$. Below we describe both settings in details and the possible use cases for both.

6.2.1 Substitute training without access to the target test set

In this setting, we use the target model's predictions to come from substitute sentences input $X_b \in \mathcal{X}_b$ instead of using the substitute dataset labels $\mathcal{Y}_b$. We use the prediction probabilities from F_{target} for the substitute training, where the loss function for training both F_b and $\mathcal{E}_\theta$ becomes:

$$\mathcal{L}_{\text{subs\_domain}}\left(F_b, Y_{\text{target}}\right) = \mathop{\mathbb{E}}_{X_b \in D_b}\left[\left\|F_{b_\phi}(\mathcal{E}_\theta(X_b)) - F_{\text{target}}(X_b)\right\|_2^2\right], \tag{6.1}$$

where D_b is the substitute dataset, $F_b\left(\mathcal{E}_\theta\left(\cdot\right)\right)$ is the substitute classifier output, and Y_{target} is the output of the target classifier F_{target}, both containing the probabilities for all available classes. This loss is the mean square error between the substitute classifier and the target classifier outputs. The purpose of this loss function is to train the substitute classifier such that its real-valued output predictions (the classes probabilities) are optimized to resemble the target classifier prediction outputs. Therefore, the selector network is trained to help the substitute classifier resemble the target classifier behavior. Fig. (6.1a) shows the substitute training procedure under this setting. This settings is suitable when the target test set in not known for the attacker, which is the common scenario of black-box attacks on pretrained models.

6.2.2 Substitute training with access to the target test set

In this setting, we use the target model's outputs to come from a part of the target test set $X_t^{\text{test}} \in \mathcal{X}_t^{\text{test}}$. The rationale behind this setting is that we consider the cases when the attacker happens to have access to the target dataset, or that the user of the target model gets to choose the training dataset, yet does not have access to the final target model

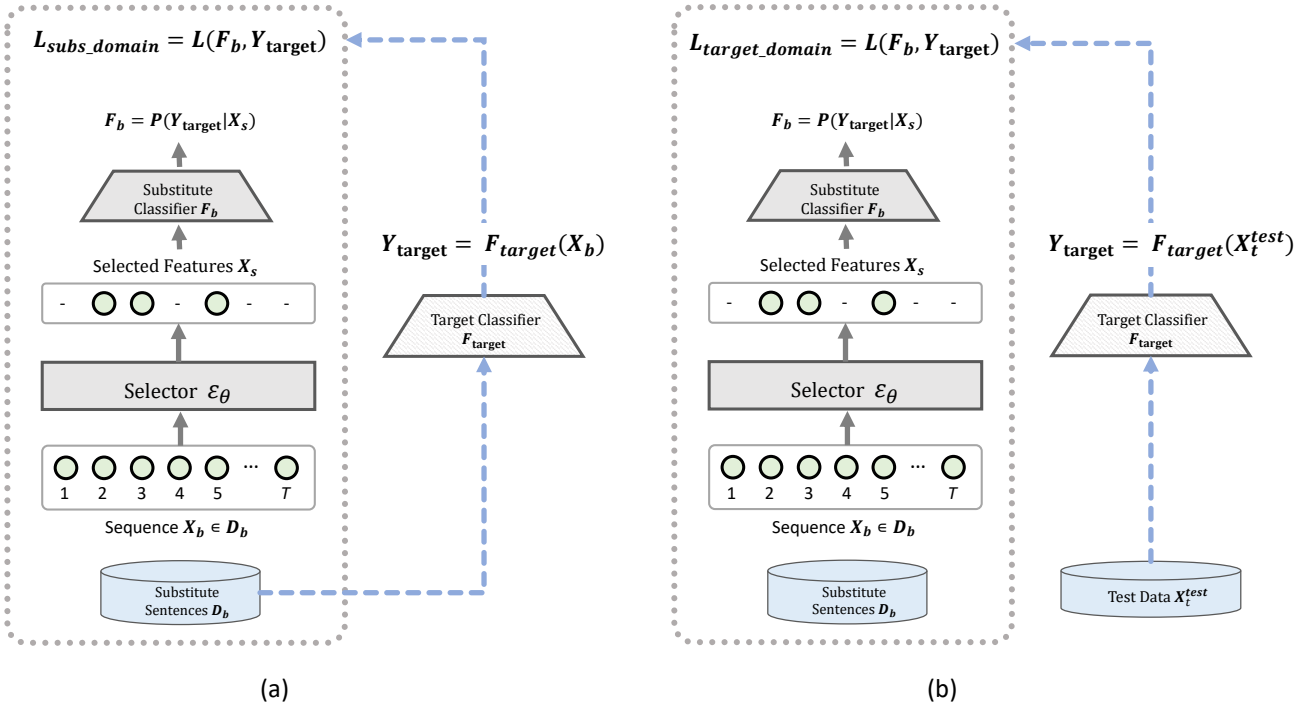

Figure 6.1: Overview of Explain2Attack L2X training with Target Guidance via a) the substitute sentences, or b) the target test set.

parameters. Similar to the previous setting, we use the probability outputs from F_{target} for the substitute training, where the loss function becomes:

$$\mathcal{L}_{\text{target\_domain}}\left(F_b, Y_{\text{target}}\right) = \underset{X_b \in D_b,\, X_t^{\text{test}} \in \mathcal{X}_t^{\text{test}}}{\mathbb{E}} \left[\left\| F_{b_\phi}(\mathcal{E}_\theta(X_b)) - F_{\text{target}}\left(X_t^{\text{test}}\right) \right\|_2^2 \right]. \tag{6.2}$$

Fig. (6.1b) shows the substitute training procedure under this setting.

6.3 Experiments and Discussion

We apply the new approach to employ target classifier predictions on sentiment classification task using WordCNN and WordLSTM target classifiers. For all of the experiments, we used the same datasets, substitute datasets, and target classifier parameters and hyperparameters that were used in Chapter 5.

To evaluate the performance of the two proposed methods described above, we report the adversarial accuracy (Adv_Acc), the average number of queries (Adv_Queries), and query efficiency, and compare with the original Explain2Attack reported in Chapter 5. In detail,

in tables 6.1 and 6.2 we report the results for incorporating the target model predictions for training Explain2Attack selector without access to the target test set and with access to it, respectively. For both Adv_Acc and Adv_Queries, the lower number is the better, indicated by the ↓ symbol, while for Query Efficiency, the higher ↑ is the better .

For the first method, we use the exact setup as in section 5.5 for the experiments, except that the substitute labels $\mathcal{Y}_b$ were replaced with the target classifier predictions from the substitute sentences, and Eq. (6.1) was used for training the selector. For the second method, $\mathcal{Y}_b$ were replaced with the target classifier predictions from the target test set, and Eq. (6.2) was used to train the selector. For the set of experiments in Table 6.2, only a small portion of the test set was used during the selector training, where 5000 out of 25,000 sentences were used.

For the set of experiments in Table 6.1, the first method was used to attack the target classifier. We can see that in three out of four experiments, that the attack rate was improved compared to the original baseline. Although the improvement is generally marginal, yet it demonstrates the added benefit of incorporating the target model predictions in the training process. Moreover, we find that the average number of queries needed was also improved in these three cases, and consequently, the query efficiency was improved.

Table 6.1: Performance metrics for Explain2Attack selector trained with target model predictions given substitute dataset sentences.

Classifier		WordCNN		WordLSTM	
Target Model		IMDB	Amazon MR	IMDB	Amazon MR
	Clean_Acc.	87.32	90.16	88.78	91.44
	(Substitute Data)	*(Yelp)*	*(IMDB)*	*(Amazon MR)*	*(IMDB)*
Adv_Acc. ↓	Explain2Attack	0.59	4.12	**0.05**	2.51
	Explain2Attack-$F_{\text{target}}(X_b)$	**0.57**	**4.11**	0.08	**2.37**
Avg_Queries ↓	Explain2Attack	402.5	351.7	**440.2**	368.7
	Explain2Attack-$F_{\text{target}}(X_b)$	**345.6**	**337.8**	652.7	**336.0**
Query Efficiency (QE) ↑	Explain2Attack	0.215	0.245	0.202	0.241
	Explain2Attack-$F_{\text{target}}(X_b)$	**0.251**	**0.255**	0.136	**0.265**

The improvement of the average number of queries is of particular interest in our setting, as it relates to the quality of word importance ranking learned during substitute training. Specifically, the baselines TextFooler (Jin et al., 2019) and the original Explain2Attack (Section 5.4) both perform the following procedure for generating an adversarial example; they perturb word by word, in order, from the more to less important words (as described

Table 6.2: Performance metrics for the selector trained with target predictions given target test set sentences (5K out of 25K target test sentences used).

Classifier		WordCNN		WordLSTM	
Target Model		IMDB	Amazon MR	IMDB	Amazon MR
	Clean_Acc.	87.32	90.16	88.78	91.44
	(Substitute Data)	*(Yelp)*	*(IMDB)*	*(Amazon MR)*	*(IMDB)*
Adv_Acc. ↓	Explain2Attack	0.59	4.12	0.05	2.51
	Explain2Attack-$F_{\text{target}}\left(X_t^{\text{test}}\right)$	**0.56**	**4.05**	0.05	**2.34**
Avg_Queries ↓	Explain2Attack	402.5	351.7	**440.2**	368.7
	Explain2Attack-$F_{\text{target}}\left(X_t^{\text{test}}\right)$	**382.6**	352.2	444.4	**357.2**
Query Efficiency (QE) ↑	Explain2Attack	0.215	0.245	0.202	0.241
	Explain2Attack-$F_{\text{target}}\left(X_t^{\text{test}}\right)$	**0.227**	0.244	0.200	**0.249**

in Algorithm 6). In each of these perturbations, the target classifier is queried to check if its prediction label was changed. Therefore, the correctness of word ranking plays a key role in the total number of words that need to be perturbed, and consequently, the total number of queries. This means that the more important the selected words are to the target classifier, the fewer total words will be needed to be perturbed in order to change the final classification label. Thus, fewer queries will be needed. By looking on the results in tables 6.1 and 6.2, we can see that for most of the experiments, there is a consistent improvement in the number of queries. This observation suggests that the employing target model predictions encouraged the selector to learn more accurate word ranking according to its importance to the target classifier. Although there is some overhead of queries involved in training the selector in the first place, the later reduction in average queries needed per single attack might be of more value with the increased number of attacks.

In addition, we find that the overhead needed for substitute training queries can be reduced in the second method, where the target test set is used for output predictions from the target classifier. As we see in Table 6.2, the selector was trained only on 5000 test set sentences, out of the total 25,000 sentences in the test set. This also suggests that having access to the target test set might have an additional benefit on both attack rates and the number of queries.

Under this particular setting of the target test set availability, we study the effect of the target set portion size used for the selector training, using 5000, 10,000, 15,000, and 20,000 samples out of the whole test set size of 25,000 samples. We perform the same set of experiments in Table 6.2 on these portions sizes and report the adversarial accuracies and average queries for all combinations of target models and datasets in Figures (6.2) and (6.3)

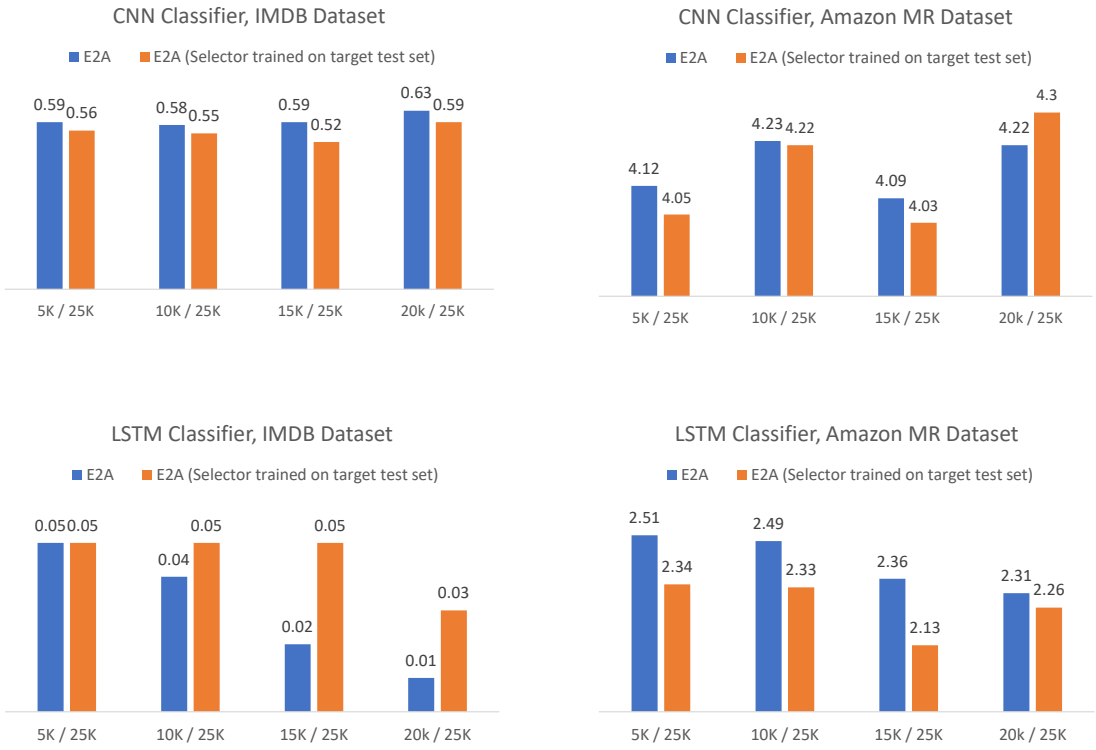

Figure 6.2: Adversarial Accuracies for Explain2Attack with the selector trained on a portion of the target test set. Each category represents the portion of the data used out of 25K samples. All combinations of target models and datasets are reported (*lower is better* ↓).

compared to the original Expalin2Attack. For every target model and dataset combination, we can see that there is an improvement either in the adversarial accuracy (lower accuracy), or in the average number of queries (fewer queries), or both. Notably, we find that the best improvements happen mostly happen when the number of used samples is less than or equal to 15,000. This suggests either that there is a limit to the number of target test set samples that can be useful for the selector training, or that this behavior is just a special case on these selected datasets, and there might be a more general consistent behavior under other datasets with different sizes.

Similarly, we believe that the number of substitute dataset samples used in the first method for training the selector might have an impact on both attack rates and the number of queries. We look further to investigate the impact of both target and substitute dataset sizes on the overall performance by choosing more datasets with different sizes, yet we leave this comprehensive study for future work.

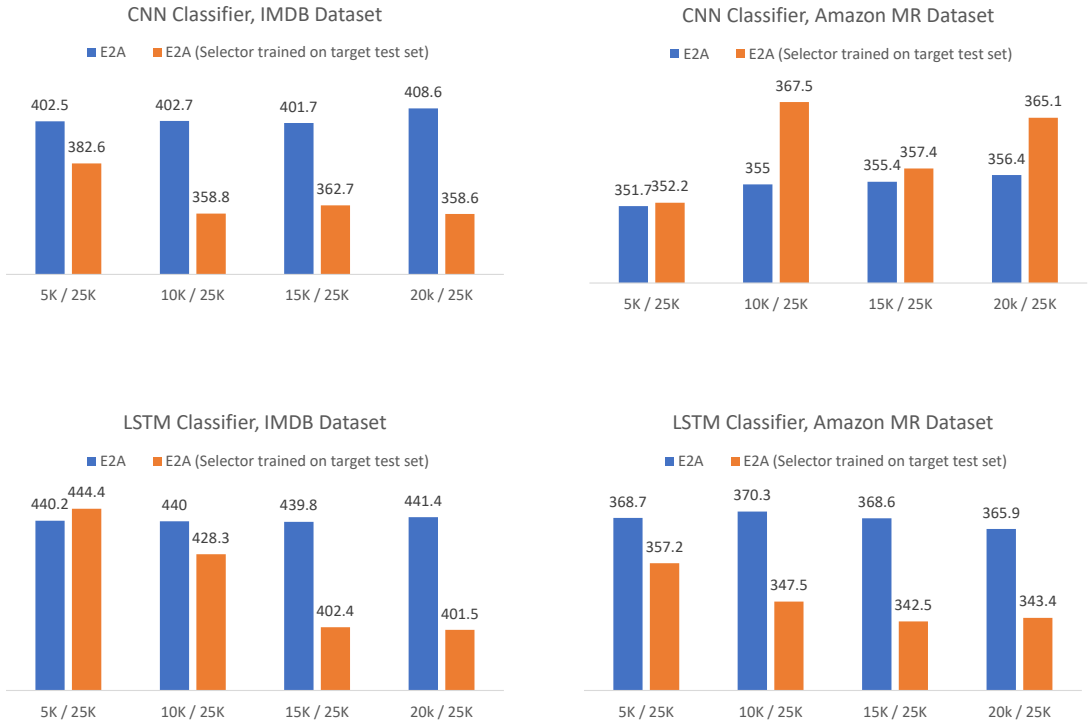

Figure 6.3: Average Queries for Explain2Attack with the selector trained on a portion of the target test set. Each category represents the portion of the data used out of 25K samples. All combinations of target models and datasets are reported (*lower is better* ↓).

6.4 Conclusion

In this chapter, we presented two methods to incorporate the target classifier output information in the process of training Explain2Attack's substitute model. Specifically, we proposed to replace the substitute labels from the original Explain2Attack with target classifier outputs during the substitute selector network training. The intuition is to encourage the selector network to better learn the behaviour of the target classifier through its outputs, thus learning more accurate word importance ranking. The experiments show that on most of the selected datasets, there was an improvement in both attack rates and the average number of queries per attack.

This study represents an initial step towards improving natural language adversarial attacks in terms of efficiency and attack rates. In the future, we plan to *i)* further leverage the target classifier information by augmenting the substitute training set with the intermediate adversarial candidates and their target classifier outputs, and *ii)* investigate the effect of mixing different substitute and target datasets and their sizes on the overall performance.

Chapter 7

Conclusions and Future Directions

7.1 Contributions

The fundamental aim of this thesis is to advance knowledge in sequence generative models and adversarial examples for sequential models. To achieve this goal, it has contributed novel models and algorithms with comprehensive empirical studies to three different tasks; *diverse sequence generation*, *controlled sequence generation*, and *natural language adversarial attacks*. First, for diverse sequence generation, we developed a principled GAN-based generative model to overcome the mode-collapse issue that causes loss of output diversity, an issue that was not addressed in a principled approach before in unsupervised GAN-based sequence models. Secondly, we proposed a novel unsupervised generative model for goal-oriented sequence generation, where the user can specify desired scores or measures to be maximized by the generative model. Lastly, in order to overcome the high complexity of generating adversarial attacks on natural language models, we proposed a novel method using interpretable substitutes across domains. The contributions of each part as well as potential future directions are presented below.

In the first part of the thesis, we presented a sequence generative framework, Adversarial Auto-regressive Networks (ARN), to address the problem of low output diversity in GANs. The motivation behind this is that for sequence generative models to become useful in real-world applications, it needs to capture the underlying data distribution accurately enough in order to generate diverse outputs. We proposed to achieve this by changing the training objective of standard GAN to combine the standard GAN "mode-seeking" behaviour

with maximum likelihood estimation (MLE) "mode-covering" behaviour. Specifically, we minimize KL divergence between data and model distributions, and minimize the Jensen-Shanon divergence, simultaneously. We theoretically prove that ARN approximates the true underlying data distribution.

In a natural language generation task, we conducted experiments to compare ARN to SeqGAN, a well-known sequence generative model. We empirically showed that ARN generated grammatically and semantically meaningful sentences, and outperformed SeqGAN in diversity and feature coverage scores. Moreover, unlike many existing unsupervised sequence generation GANs like SeqGAN, ARN incorporates a compressed latent space representation, which is learned by maximizing a variational lower bound on the first token. This capability makes ARN highly useful for applications where high level representations from the data can be learned. These learned representations can then be used for controlling/guiding the generation process through changing/fixing specific factors of the latent space according to the desired outcomes.

In the second part of the thesis, we proposed OptiGAN, a novel GAN-based model for controlled sequence generation that employs reinforcement learning as a control mechanism to maximize desired domain rewards. OptiGAN is specifically useful for goal-oriented generation applications, where the model is required to maximize certain goals, scores or metrics for its generated output (e.g. generating a chemical molecule with high solubility, activity and ease of synthesis). This type of generative models can be used in different real-world applications, including autonomous motion planning and drug or material design.

OptiGAN combines the diversity encouraged GAN framework in (ARN) with Reinforcement Learning (RL), since using RL methods like policy gradients allows maximizing accumulative future rewards. The final generator objective is a combined MLE and GAN optimisation, and maximisation of the future rewards using REINFORCE policy gradients. To the best of our knowledge, OptiGAN is the first GAN-based controlled generative model for sequences that addresses the diversity issue in a principled approach, where it combines the benefits of GAN and RL policy learning, while avoiding their mode-collapse and high variance drawbacks.

By conducting comprehensive experiments in two different tasks; text generation and aircraft trajectory generation, we evaluate both the diversity and optimized scores of generated outputs from OptiGAN against other sequence generative models. We found that OptiGAN

achieves higher scores than baseline models based on desired scores to be maximized, while preserving the diversity of generated outputs. Additionally, we empirically show that if only pure RL is applied to maximize a score of interest, that the realism of the output might be sacrificed for the sake of superficially obtaining higher scores. Hence we conclude that combining a GAN-based objective with RL encourages the optimisation procedure of RL to stay close to the true data distribution.

In the third part of the thesis, we proposed a novel adversarial attacks framework, *Explain2Attack*, an efficient framework to generate black-box adversarial attacks on natural language models. In a typical black-box setting, the main challenge we address is the query and computation cost of adversarial sentences, where the number of queries is of critical concern for an attacking agent. Most of the existing methods depend on heavily querying the target model to choose which words to perturb with replacement synonyms. Instead, we proposed a novel approach to address this problem through interpretability across domains, in order to reduce the number of queries needed, and increase the computational efficiency. The main intuition is to learn word ranking that most probably impacts the target model instead of searching for it expensively. Specifically, we train an interpretable substitute model on a substitute dataset, where the substitute model learns word importance scores for the word ranking on the substitute domain. The substitute model is then used in the word ranking step in the target domain to generate word importance scores.

To show the efficiency of Explain2Attack, we conducted extensive experiments and introduced two new evaluation metrics to measure the practical advantage of query reduction compared to the attack rate. Our method is able to achieve or out-perform state-of-the-art methods in attack rates, yet with reduced number of queries. Another key advantage of our framework is that it significantly scales better with the input length, since it does not perform word by word ranking. In Chapter 6, we further proposed to improve the attack rates and number of queries by employing target model output predictions in the substitute training, and validated with a preliminary set of experiments the improvements in both attack rates and the average number of queries. To the best of our knowledge, Explain2Attack is the first framework to learn word importance scores, instead of searching for it using traditional expensive procedures. Our framework advances the state-of-the-art in this area, where expensive word ranking procedures are not needed any more to generate successful adversarial attacks.

7.2 Future Directions

This dissertation broadly covers many topics in deep sequence modeling, with broad impact on this crucial research field, and wider impact on many related applications. There are many potential future directions that could be pursued, among which two promising directions are discussed below.

The first direction is in goal-oriented controlled sequence generation via disentangled representations. By connecting the latent representations capacity in Chapter 3 with goal-oriented generation in Chapter 4, an advanced controlled generative model could be developed. In the existing literature, controlled generation is either addressed through factorized disentangled latent space, or through reward optimisation mechanism. However, it is not addressed through both approaches in the same model. A disentangled generative model could be realized by disentangling the latent space into distinct meaningful latent sub-spaces in an unsupervised or semi-supervised way, where each sub-space is responsible for a certain attribute or factor of variation from the training data (Khemakhem et al., 2020). Given this disentangled latent space, an advanced version of our model in Chapter 4 can then address controlled generation in a unified disentangled goal-oriented framework. An example use-case would be that the user wants to generate a sequence that contains a certain attribute, yet achieves the best score on another factor of variation (e.g. to generate a specific style of classical music (Mozart or Bach style) yet with highly happy rhythm). Therefore, if this capability is achieved, it could have a significant impact on certain real-world applications and industries, including chemical engineering, drug discovery, and advanced manufacturing.

The second direction is direct cost-efficient black-box adversarial attacks on natural language, where the query cost and attack rates can be further improved in different ways. First, synonym ranking and replacement procedure in our method and other existing methods requires significant number of queries to the target model. This part is responsible for the most queries needed for a successful attack. Therefore, a potential solution to this problem it to choose replacement synonyms through a pretrained language model (Li et al., 2020). Moreover, our word ranking learning through a substitute model can be further fine-tuned to directly attack the target model. Building on the initial work presented in Chapter 6, the substitute training and word ranking might be further improved by utilizing the intermediate perturbed sentences during adversarial candidate search, and using them to augment

the substitute training set. Given these enhancements, black-box adversarial models could move towards less dependence on target model probability output during attacks, with only initial substitute training or intermediate word replacement decisions remain dependent on target outputs predictions. More importantly, these improved attack methods could be used to augment downstream models with adversarial examples as training data to improve its robustness, which is one of the key goals of generating adversarial examples.

Appendices

A Final objective function in Eq. (3.5)

Proof. Consider this optimisation problem:

$$\max_{G}\min_{D}\Big[\mathbb{E}_{X\sim p_d}\left[\log\, p_G\left(X\mid\theta\right)\right]- \tag{1}$$
$$\mathbb{E}_{X\sim p_d}\left[\log\, D\left(X\right)\right]-\mathbb{E}_{z\sim p_z}\left[\log\left[1-D\left(G\left(z\right)\right)\right]\right]\Big].$$

Given a generator G, the optimal $D^*\left(G\right)$ is determined as:

$$D^*_G\left(X\right)=\frac{p_d\left(X\right)}{p_G\left(X\right)+p_d\left(X\right)},$$

where $p_G\left(X\right)$ is the distribution induced from $G\left(X\right)$ where $X\sim p_d\left(X\right)$.

Substituting D^*_G back to Eq. (1), we obtain the following optimisation problem regarding G:

$$\max_{G}\left(\mathbb{E}_{p_d}\left[\log\, p_G\left(X\right)\right]-JS\left(P_d\|P_G\right)\right). \tag{2}$$

The objective function in Eq. (2) can be written as

$$\begin{aligned}&\mathbb{E}_{p_d}\left[\log\, p_G\left(X\right)\right]-JS\left(P_d\|P_G\right)\\ =&-JS\left(P_d\|P_G\right)-KL\left(P_d\|P_G\right)-\mathbb{E}_{p_d}\left[\log\, p_d\left(X\right)\right]\\ =&-JS\left(P_d\|P_G\right)-KL\left(P_d\|P_G\right)+\text{const.}\end{aligned}$$

Therefore, the optimisation problem in Eq. (2) is equivalent to:

$$\min_{G}\left[JS\left(P_d\|P_G\right)+KL\left(P_d\|P_G\right)\right].$$

At the Nash equilibrium point of this game, we hence obtain: $p_G(X) = p_d(X)$. □

B Policy Gradient Theorem

Proof. The probability $p(\tau \mid \theta)$ of a trajectory $\tau = (S_0, A_0, ..., S_{T-1}, A_{T-1}, R_T)$ given that actions come from π_θ is expressed as:

$$p(\tau \mid \theta) = \rho_0(S_0) \prod_{t=0}^{T-1} P(S_{t+1} \mid S_t, A_t) \pi_\theta(A_t \mid S_t). \tag{3}$$

We can derive the basic policy gradients update starting from Eq. (2.10) as:

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [G_t(\tau)] \\
&= \nabla_\theta \int p(\tau \mid \theta) G_t(\tau)\, d\tau \\
&= \int \nabla_\theta (p(\tau \mid \theta) G_t(\tau))\, d\tau \\
&= \int (G_t(\tau) \nabla_\theta p(\tau \mid \theta) + p(\tau \mid \theta) \nabla_\theta G_t(\tau))\, d\tau && \text{chain rule} \\
&= \int G_t(\tau) \nabla_\theta p(\tau \mid \theta)\, d\tau && \nabla_\theta G_t(\tau) = 0 \\
&= \int G_t(\tau) p(\tau \mid \theta) \frac{\nabla_\theta p(\tau \mid \theta)}{p(\tau \mid \theta)}\, d\tau \\
&= \int G_t(\tau) p(\tau \mid \theta) \nabla_\theta \log p(\tau \mid \theta)\, d\tau \\
&= \int G_t(\tau) p(\tau \mid \theta) \nabla_\theta \left[\log \rho_0(S_0) + \sum_{t=0}^{T-1} \left(\log P(S_{t+1} | S_t, A_t) + \log \pi_\theta(A_t | S_t) \right) \right] d\tau && \text{using (3)} \\
&= \int G_t(\tau) p(\tau \mid \theta) \left[\cancel{\nabla_\theta \log \rho_0(S_0)} + \sum_{t=0}^{T-1} \left(\cancel{\nabla_\theta \log P(S_{t+1} | S_t, A_t)} + \nabla_\theta \log \pi_\theta(A_t | S_t) \right) \right] d\tau && \nabla_\theta \rho_0, \nabla_\theta P = 0 \\
&= \int G_t(\tau) p(\tau \mid \theta) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t | S_t)\, d\tau \\
&= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[G_t(\tau) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t | S_t) \right],
\end{aligned}$$

where $G_t(\tau)$, ρ_0 and P do not depend on θ. □

C The McGrew Score

Contact Range

The relative 3D Euclidean distance/range in meters between aircraft 1 and 2:

$$R_{12} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2},$$

where that $R12 = R21$.

Aspect Angle and Antenna Train Angle

As shown in Figure 1, Aspect Angle (AA) is the angle of aircraft 1 relative to aircraft 2's tail and Antenna Train Angle (ATA), also known as Bearing Angle (BA) is the angle aircraft 2 makes relative to the aircraft's nose.

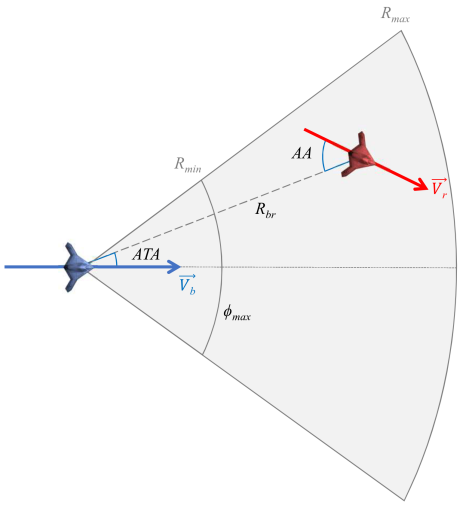

Figure 1: Relative angle definitions Aspect angle AA and Antenna Train Angle ATA between Blue and Red

McGrew Scoring

McGrew (McGrew et al., 2010) used a specific system to keep score of how well an aircraft was doing relative to another aircraft in an attempt to get behind the other aircraft. McGrew's scoring system has two components; an angular component and a range component. It makes use of the range R_{12} as well as the aspect angle AA and antenna train angle ATA to calculate the score. It also makes use of a number of adjustable hyper parameters.

McGrew's Angular Score A_M

McGrew's angular score is given by the following equation where AA and ATA are specified in degrees.

$$A_M = \frac{1}{2}\left[\left(1 - \frac{AA}{180^\circ}\right) + \left(1 - \frac{ATA}{180^\circ}\right)\right].$$

Note that in the ideal case where $AA = 0$ and $ATA = 0$, the maximum angular score is $M_A = 1$.

McGrew Range Score R_M

The second component of McGrew's scoring system is the range score. The range is scored is defined as follows:

$$R_M = \exp\left(-\frac{|R - R_d|}{k \times 180^\circ}\right),$$

where R is current range between the two aircraft, R_d is the desired range and k is a hyper-parameter scaling factor. This means that the score is range dependent and is highest when it is closest to the desired range R_d . The values of R_d and k need to be determined for the exact application.

The value of k determines how wide the function peak around R_d is. The larger the k the larger the spread. The smaller the value of k the narrower the peak and hence a higher score can only be achieved by being very close to the desired range. In ACE Zero the default value of $k = 5$ is used.

McGrew Score S_M

Finally, the total SM is given by the range score multiplied by the angular score:

$$S_M = A_M R_M = \frac{1}{2}\left[\left(1 - \frac{AA}{180^\circ}\right) + \left(1 - \frac{ATA}{180^\circ}\right)\right] \exp\left(-\frac{|R - R_d|}{k \times 180^\circ}\right).$$

Bibliography

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1316. URL `https://www.aclweb.org/anthology/D18-1316`.

Fred Austin, Giro Carbone, Hans Hinz, Michael Lewis, and Michael Falco. Game theory for automated maneuvering during air-to-air combat. *Journal of Guidance, Control, and Dynamics*, 13(6):1143–1149, 1990.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

Yonatan Belinkov and James R. Glass. Analysis methods in neural language processing: A survey. *CoRR*, abs/1812.08951, 2018. URL `http://arxiv.org/abs/1812.08951`.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the Twentieth Conference on Computational Natural Language Learning (CoNLL)*., 2016.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high

fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1xsqj09Fm`.

Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 883–892. PMLR, 10–15 Jul 2018a.

Liqun Chen, Shuyang Dai, Chenyang Tao, Dinghan Shen, Zhe Gan, Haichao Zhang, Yizhe Zhang, and Lawrence Carin. Adversarial text generation via feature-mover's distance. In *NIPS*, 2018b.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2980–2988. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5653-a-recurrent-latent-variable-model-for-sequential-data.pdf`.

N. Dam, Q. Hoang, T. Le, T. D. Nguyen, H. Bui, and D. Phung. Three-player Wasserstein GAN via amortised duality. In *International Joint Conference on Artificial Intelligence*, 2019.

Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

Ambra Demontis, Marco Melis, Maura Pintor, Jagielski Matthew, Battista Biggio, Oprea Alina, Nita-Rotaru Cristina, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, volume 28. {USENIX} Association, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. GANSynth: Adversarial neural audio synthesis. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1xQVn09FX`.

William Fedus, Ian Goodfellow, and Andrew M. Dai. MaskGAN: Better text generation via filling in the. In *International Conference on Learning Representations*, pages 1–15, 2018. URL `https://openreview.net/forum?id=ByOExmWAb`.

Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pages 2199–2207, 2016.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56. IEEE, 2018.

Siddhant Garg and Goutham Ramakrishnan. Bae: Bert-based adversarial examples for text classification, 2020.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques for Building Intelligent Systems*. O'Reilly Media, 2017. ISBN 9781491962299. URL `https://books.google.com.au/books?id=I6qkDAEACAAJ`.

Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, pages 1–57, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014a.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014b.

Alex Graves. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer, 2012.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, pages 1–43, 2013.

Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and AlÃ¡n Aspuru-Guzik. Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models, 2017.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Junxian He, Daniel Spokoyny, Graham Neubig, and Taylor Berg-Kirkpatrick. Lagging inference networks and posterior collapse in variational autoencoders. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rylDfnCqF7`.

Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Phung. MGAN: Training generative adversarial nets with multiple generators. In *International Conference on Learning Representations*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, J Jurgen Schmidhuber, and Corso Elvezia. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. pages 1–15.

K Hornik, M Stinchcombe, and H White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Mahmoud Hossam, Trung Le, Michael Papasimeon, Viet Huynh, and Dinh Phung. Text generation with deep variational GAN. In *Third workshop on Bayesian Deep Learning (NeurIPS 2018)*, Montreal, Canada, 2018.

Mahmoud Hossam, Trung Le, Viet Huynh, Michael Papasimeon, and Dinh Phung. OptiGAN: Generative adversarial networks for goal optimized sequence generation, 2020.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Toward controlled generation of text. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1587–1596, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL `http://proceedings.mlr.press/v70/hu17e.html`.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. *ArXiv e-prints*, abs/1611.01144, 2016a. URL `http://arxiv.org/abs/1611.01144`.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016b.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is BERT really robust? natural language attack on text classification and entailment. *CoRR*, abs/1907.11932, 2019. URL `http://arxiv.org/abs/1907.11932`.

Ilyes Khemakhem, Diederik Kingma, Ricardo Monti, and Aapo Hyvarinen. Variational autoencoders and nonlinear ica: A unifying framework. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2207–2217, Online, 26–28 Aug 2020. PMLR. URL `http://proceedings.mlr.press/v108/khemakhem20a.html`.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *The 2nd International Conference on Learning Representations (ICLR)*, 2013.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL `http://arxiv.org/abs/1312.6114`.

Volodymyr Kuleshov, Shantanu Thakoor, Tingfung Lau, and Stefano Ermon. Adversarial examples for natural language classification problems, 2018.

Matt J Kusner and José Miguel Hernández-Lobato. GANs for sequences of discrete elements with the gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016.

Hung Le, Truyen Tran, Thin Nguyen, and Svetha Venkatesh. Variational memory encoder-decoder. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1508–1518. Curran Associates, Inc., 2018a. URL `http://papers.nips.cc/paper/7424-variational-memory-encoder-decoder.pdf`.

T. Le, H. Vu, T. Nguyen, and D. Phung. Geometric enclosing networks. In *In Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, 2018b.

T. Le, Q. Hoang, H. Vu, T D. Nguyen, H. Bui, and D. Phung. Learning generative adversarial networks from multiple data sources. In *International Joint Conference on Artificial Intelligence*, 2019.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. Bert-attack: Adversarial attack against bert using bert, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Andrew Maas. Large movie review dataset, 2011. URL `https://ai.stanford.edu/~amaas/data/sentiment/`.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011a.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 142–150, 2011b. URL `http://www.aclweb.org/anthology/P11-1015`.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *ArXiv e-prints*, abs/1611.00712, 2016a. URL `http://arxiv.org/abs/1611.00712`.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The CONCRETE distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016b.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

James S McGrew, Jonathon P How, Brian Williams, and Nicholas Roy. Air-combat strategy using approximate dynamic programming. *Journal of guidance, control, and dynamics*, 33(5):1641–1654, 2010.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, pages 1–25, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

T. Nguyen, T. Le, Hung Vu, and D. Phung. Dual discriminator generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2670–2680, 2017.

Weili Nie, Nina Narodytska, and Ankit Patel. RelGAN: Relational generative adversarial networks for text generation. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJedV3R5tm`.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, pages 1–15, 2016a.

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4797–4805. Curran Associates Inc., 2016b.

Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *arXiv preprint cs/0506075*, 2005.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings*

of the 2017 ACM on Asia conference on computer and communications security, pages 506–519, 2017.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA.*, pages 311–318, 2002. URL `http://www.aclweb.org/anthology/P02-1040.pdf`.

Daniil Polykovskiy, Alexander Zhebrak, Dmitry Vetrov, Yan Ivanenkov, Vladimir Aladinskiy, Polina Mamoshina, Marine Bozdaganyan, Alexander Aliper, Alex Zhavoronkov, and Artur Kadurin. Entangled conditional adversarial autoencoder for de novo drug discovery. *Molecular Pharmaceutics*, 15(10):4398–4405, 2018. doi: 10.1021/acs.molpharmaceut.8b00839. URL `https://doi.org/10.1021/acs.molpharmaceut.8b00839`. PMID: 30180591.

Ben Poole, Alexander A Alemi, Jascha Sohl-Dickstein, and Anelia Angelova. Improved generator objectives for gans. *arXiv preprint arXiv:1612.02780*, pages 1–8, 2016.

Evgeny Putin, Arip Asadulaev, Quentin Vanhaelen, Yan Ivanenkov, Anastasia V. Aladinskaya, Alex Aliper, and Alex Zhavoronkov. Adversarial threshold neural computer for molecular de novo design. *Molecular Pharmaceutics*, 15(10):4386–4397, 2018. doi: 10.1021/acs.molpharmaceut.7b01137. URL `https://doi.org/10.1021/acs.molpharmaceut.7b01137`. PMID: 29569445.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, pages 1–16, 2015.

Miquel Ramirez, Michael Papasimeon, Lyndon Behnke, Nir Lipovetzky, Tim Miller, and Adrian R. Pearce. Real–time uav maneuvering via automated planning in simulations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 5243–5245, 2017. doi: 10.24963/ijcai.2017/778. URL `https://doi.org/10.24963/ijcai.2017/778`.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in Neural Information Processing Systems*, pages 14837–14847, 2019.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1103.

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. *arXiv preprint arXiv:1803.05428*, 2018.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986. doi: 10.1038/323533a0.

Motoki Sato, Jun Suzuki, Hiroyuki Shindo, and Yuji Matsumoto. Interpretable adversarial perturbation in input embedding space for text. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4323–4330. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/601. URL `https://doi.org/10.24963/ijcai.2018/601`.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

J. Michael Steele. *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511817106.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Richard S. Sutton and Andrew G. Barto. Reinforcement learning, second edition, 2018.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Daniel Takeshi. Going deeper into reinforcement learning: Fundamentals of policy gradients, 2017. URL `https://danieltakeshi.github.io/2017/03/28/going-deeper-into-reinforcement-learning-fundamentals-of-policy-gradients/`.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205): 1–37, 2016.

Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Prashanth Vijayaraghavan and Deb Roy. Generating black-box adversarial examples for text classifiers using a deep reinforced model, 2019.

Wenqi Wang, Benxiao Tang, Run Wang, Lina Wang, and Aoshuang Ye. Towards a robust deep neural network in texts: A survey. *CoRR*, abs/1902.07285, 2019. URL `http://arxiv.org/abs/1902.07285`.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992.

Puyudi Yang, Jianbo Chen, Cho-Jui Hsieh, Jane-Ling Wang, and Michael I. Jordan. Greedy attack and gumbel attack: Generating adversarial examples for discrete data, 2019. URL `https://openreview.net/forum?id=ByghKiC5YX`.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. SeqGAN: Sequence generative adversarial nets with policy gradient. In *Proc. of AAAI*, pages 2852–2858, 2017.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors,

Proceedings of the 36th International Conference on Machine Learning, volume 97 of *Proceedings of Machine Learning Research*, pages 7354–7363, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/zhang19d.html`.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. Adversarial feature matching for text generation. In *ICML*, 2017.