A FRAMEWORK FOR INTELLIGENT NETWORK AND SECURITY

By

JIADE LI

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY UNIVERSITY OF FLORIDA 2017
To my family
ACKNOWLEDGMENTS

I would like to express my thanks to everyone helping me all the way when I was doing my Ph.D. study. Especially, I would like to thank my advisor Dr. Dapeng Oliver Wu who has always been considerate and helpful and always giving me kind and useful suggestions not only regarding my study, but also regarding my life and future career. In addition, I would also like to thank Dr. Tan Foon Wong, Dr. Xiaolin Andy Li and Dr. Ye Xia. As my supervisory committee members, they always provide timely and insightful suggestions regarding my Ph.D. study.
TABLE OF CONTENTS

ACKNOWLEDGMENTS ........................................................................................................... 4
LIST OF TABLES ..................................................................................................................... 8
LIST OF FIGURES .................................................................................................................. 9
ABSTRACT .................................................................................................................................. 11

CHAPTER

1 INTRODUCTION .................................................................................................................. 13

2 PRACTICAL DISTRIBUTED SCHEDULING FOR QOS-AWARE SMALL CELL MMWAVE MESH BACKHAUL NETWORK .............................................................................. 18
   2.1 Problem Formulation .................................................................................................... 18
   2.2 System Model .............................................................................................................. 19
      2.2.1 Network Model .................................................................................................. 19
      2.2.2 Directional Antenna Model .............................................................................. 20
   2.3 Problem Formulation .................................................................................................. 22
   2.4 Distributed Scheduling Algorithm ............................................................................. 24
      2.4.1 Neighborhood Detection .................................................................................... 24
      2.4.2 Contention Window Prioritization ...................................................................... 25
      2.4.3 Coarse-to-Fine Window Mapping Algorithm .................................................... 26
         2.4.3.1 Coarse phase ............................................................................................. 27
         2.4.3.2 Fine phase ................................................................................................. 28
   2.5 Simulation Results ...................................................................................................... 29
      2.5.1 Development of Simulator .................................................................................. 29
      2.5.2 Performance Evaluation ...................................................................................... 30
         2.5.2.1 Effects of number of flows on throughput performance .............................. 31
         2.5.2.2 Effects of number of packets in each flow on throughput performance ...... 33
         2.5.2.3 Effects of antenna beamwidth on throughput performance ....................... 34
         2.5.2.4 Effects of number of flows on packet loss ratio ........................................ 35
   2.6 Chapter Summary ........................................................................................................ 36

3 BI-DIRECTIONAL ATTENTION GENERATIVE ADVERSARIAL NETWORKS FOR USER PROFILE IDENTIFICATION AND ANOMALY DETECTION .............................................................................. 38
   3.1 Problem Formulation ................................................................................................ 38
   3.2 Building Feature Vectors .......................................................................................... 40
   3.3 Dimension Reduction ................................................................................................. 44
      3.3.1 Literature Review .............................................................................................. 45
      3.3.2 Autoencoders ..................................................................................................... 47
      3.3.3 Stacked Autoencoders ..................................................................................... 49
3.3.4 Design and Implementation ............................................. 51
3.4 The GAN Framework ....................................................... 53
  3.4.1 Discriminator .......................................................... 54
    3.4.1.1 Generator ...................................................... 57
    3.4.1.2 Training ....................................................... 58
  3.4.2 Cost Functions .......................................................... 58
    3.4.2.1 For discriminator ............................................. 58
    3.4.2.2 For generator ............................................... 59
    3.4.2.3 Comparison of cost functions ................................ 60
3.5 Bi-directional Attention GAN (AttenGAN) .................................. 61
  3.5.1 Sequence Encoder ...................................................... 61
  3.5.2 Sequence Attention Layer ........................................... 63
  3.5.3 Model Training .......................................................... 63
  3.5.4 Preparing Training Data ............................................... 65
3.6 Experiment Results .......................................................... 65
  3.6.1 Model Configuration .................................................... 66
  3.6.2 User Profile Identification Accuracy .................................. 66
  3.6.3 Anomaly Detection Capability ........................................ 68
  3.6.4 Convergence Speed .................................................... 70
  3.6.5 Attention Weight Distribution ....................................... 71
3.7 Chapter Summary .............................................................. 72
4 PARTIAL LEARNING GENERATIVE ADVERSARIAL NETWORKS FOR USER
PROFILE IDENTIFICATION AND ANOMALY DETECTION ....................... 74
  4.1 Problem Formulation ...................................................... 74
  4.2 Related Models ............................................................. 75
    4.2.1 Maximum Likelihood Estimation .................................. 75
    4.2.2 A Taxonomy of Deep Generative Models ......................... 77
    4.2.3 Explicit Density Models .......................................... 79
    4.2.4 Implicit Density Models .......................................... 82
    4.2.5 Comparing Generative Models .................................... 82
    4.2.6 Sequential GAN ................................................... 83
  4.3 Review of LSTM ............................................................. 85
    4.3.1 Vanilla LSTM ....................................................... 88
    4.3.2 Forward pass ........................................................ 88
    4.3.3 Backpropagation through time ..................................... 90
  4.4 Partial Learning Sequential GAN (P-GAN) ................................ 90
    4.4.1 Generative Adversarial Partial Learning ......................... 91
    4.4.2 Time Step Extension ............................................... 93
    4.4.3 Model Training ..................................................... 94
    4.4.4 Model Building ..................................................... 95
  4.5 Experiment Results ....................................................... 97
    4.5.1 Experiment Setup ................................................... 97
    4.5.2 User Profile Identification Accuracy ................................ 97

6
4.5.3 Anomaly Detection Capability ........................................ 99
4.5.4 Convergence Speed .................................................. 101
4.6 Chapter Summary ......................................................... 102

5 CONCLUSIONS ............................................................... 103
REFERENCES ................................................................. 105
BIOGRAPHICAL SKETCH .................................................... 117
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Contention Window Prioritization</td>
<td>26</td>
</tr>
<tr>
<td>3-1</td>
<td>AttenGAN accuracy comparison at Epoch 30.</td>
<td>68</td>
</tr>
<tr>
<td>4-1</td>
<td>P-GAN identification accuracy comparison at Epoch 30.</td>
<td>99</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>The mesh backhaul network in the small cells densely deployed scenario.</td>
<td>20</td>
</tr>
<tr>
<td>2-2</td>
<td>Topology of base stations used in simulation.</td>
<td>31</td>
</tr>
<tr>
<td>2-3</td>
<td>Number of flows on throughput performance.</td>
<td>32</td>
</tr>
<tr>
<td>2-4</td>
<td>Number of packets in each flow on throughput performance.</td>
<td>33</td>
</tr>
<tr>
<td>2-5</td>
<td>Antenna beamwidth on throughput performance.</td>
<td>35</td>
</tr>
<tr>
<td>2-6</td>
<td>Effects of Number of Flows on Packet Loss Ratio, the less the better.</td>
<td>36</td>
</tr>
<tr>
<td>3-1</td>
<td>File system sample log</td>
<td>42</td>
</tr>
<tr>
<td>3-2</td>
<td>Network data sample log</td>
<td>42</td>
</tr>
<tr>
<td>3-3</td>
<td>Process data sample log</td>
<td>43</td>
</tr>
<tr>
<td>3-4</td>
<td>Structure of a typical autoencoder</td>
<td>48</td>
</tr>
<tr>
<td>3-5</td>
<td>Illustration of an autoencoder framework</td>
<td>49</td>
</tr>
<tr>
<td>3-6</td>
<td>Illustration of a stacked autoencoder framework</td>
<td>50</td>
</tr>
<tr>
<td>3-7</td>
<td>Implementation of a stacked autoencoder for our dataset</td>
<td>52</td>
</tr>
<tr>
<td>3-8</td>
<td>Illustration of the GAN framework</td>
<td>55</td>
</tr>
<tr>
<td>3-9</td>
<td>The graphical model structure of GANs</td>
<td>56</td>
</tr>
<tr>
<td>3-10</td>
<td>Illustration of AttenGAN model framework</td>
<td>62</td>
</tr>
<tr>
<td>3-11</td>
<td>AttenGAN identification accuracy compared with other schemes.</td>
<td>67</td>
</tr>
<tr>
<td>3-12</td>
<td>AttenGAN ROC curve compared with other schemes.</td>
<td>69</td>
</tr>
<tr>
<td>3-13</td>
<td>AttenGAN convergence speed comparison with other schemes.</td>
<td>70</td>
</tr>
<tr>
<td>3-14</td>
<td>AttenGAN attention weight distribution for 3 profiles.</td>
<td>72</td>
</tr>
<tr>
<td>4-1</td>
<td>The maximum likelihood process: $\theta^* = \arg\max_{\theta} \mathbb{E}<em>{x \sim \text{data}} \log p</em>{\text{model}}(x</td>
<td>\theta)$</td>
</tr>
<tr>
<td>4-2</td>
<td>Deep generative models that can learn via the principle of maximim likelihood differ with respect to how they represent or approximate the likelihood.</td>
<td>78</td>
</tr>
<tr>
<td>4-3</td>
<td>Detailed schematic of the SRN unit (left) and an LSTM block (right) as used in the hidden layers of a recurrent neural network.</td>
<td>87</td>
</tr>
<tr>
<td>4-4</td>
<td>Illustration of our proposed P-GAN training framework.</td>
<td>91</td>
</tr>
</tbody>
</table>
4-5 P-GAN identification accuracy compared with other schemes. .......................... 98
4-6 P-GAN ROC curve compared with other schemes. ............................................ 100
4-7 P-GAN convergence speed compared with other schemes. ............................... 102
This dissertation presents my work to build a framework for high efficient and high throughput network and the relevant network security problems. With the explosive growth of mobile data demand, small cells densely deployed underlying the homogeneous macrocells are emerging as a promising candidate for the fifth generation (5G) mobile network. The backhaul communication for small cells poses a significant challenge. However, with huge bandwidth available in the mmWave band, the wireless backhaul at mmWave frequencies can be a promising backhaul solution for small cells. In this work, we propose Distributed Maximum QoS-aware (DMQ) scheduling algorithm for the mmWave backhaul network of small cells to maximize the system throughput while satisfying QoS requirements for each flow. Based on CSMA/CA, the proposed algorithm is the first that prioritizes MAC contention window to provide better concurrent transmission support while achieving QoS-aware capability. Simulations in the 73 GHz band are conducted to demonstrate 25%–40% performance boost in terms of system throughput and packet loss ratio against other existing schemes.

As we are entering the information age, more and more network and data security problems are arising. The current state-of-the-art in computer infrastructure defense solutions focus on one layer of operation with deployments coming in a “one size fits all” format, without taking into account the unique way people use their computers. We are facing the challenge that personalized usage profiles are proliferating and each user profile poses different threat levels and types. Based on a paradigm of cross-layer personalization security model, we
solve this problem by analyzing user profile anomalies. Considering the sequential nature of our collected user profile data, we propose a compound framework to achieve user profile anomaly detections. The collected user profile will go through modules including hashing, dimension reduction and classification. The dimension reduction is built upon idea of autoencoder and we propose two schemes for classification. One introduces attention mechanism to sequential generative adversarial nets (GAN) and the other brings partial sequence learning mechanism to GAN. We show both schemes give good results and achieve high classification accuracy and anomaly detection capability.
CHAPTER 1
INTRODUCTION

Some industry and academic experts predict a 1000-fold demand of mobile data increase by 2020 [1]. In order to offer such gigantic demand in data rates and throughput, small cells densely deployed underlying the conventional homogeneous macrocells are emerging as a promising candidate for the fifth generation (5G) mobile broadband [2]. This new network deployment is usually referred to as heterogeneous cellular networks (HCNs). However, with the increase of the number of small cells deployed, the backhaul for small cells becomes a significant challenge [2–5]. Although fiber based backhaul offers large bandwidth, it is costly, inflexible, and time-consuming to connect the densely deployed small cells. In contrast, wireless backhaul is more cost-effective, flexible, and easier to deploy [3]. With huge bandwidth available, wireless backhaul in mmWave bands, such as the 60 GHz band and E-band (71–76 GHz and 81–86 GHz), provides several-Gbps data rates and can be a promising backhaul solution for small cells. Meanwhile, rapid progress in mmWave circuits and antennas has accelerated popularization of wireless products and services in mmWave bands [6]. There are also several standards defined in the mmWave bands, such as ECMA-387 [7], IEEE 802.15.3c [8], and IEEE 802.11ad [9].

In order to solve the problem and reduce system complexity, there are a few emerging works that propose applying CSMA contention based scheduling on 60Ghz mmWave band. CSMA has been widely adopted in various existing 802.11 standards and is a practical distributed scheduling framework that does not need centralized control. Gong et al. [10] propose a directional CSMA/CA protocol designed specifically for 60GHz WPANs. It adopts virtual carrier sensing and relies on a central coordinator to distribute network allocation vector (NAV) information. The authors also extended the work to support spatial reuse in [11]. Similar works can also be found in [12]. Another notable work by Zheng et al. [13] proposes an optimal scheduling algorithm for contention based network although it does not specifically target on mmWave band. However, all these approaches only target on channel access control...
and are generally lack of QoS support. Thus they cannot be applied to QoS-aware backhaul networks.

In this dissertation, we develop a practical CSMA contention based scheduling scheme that is also QoS aware for the small cell backhaul network specifically in mmWave band. The proposed scheme operates in a completely distributed fashion and does not need centralized controller to coordinate flow scheduling. To the best of our knowledge, this is the first CSMA contention based scheduling protocol that prioritizes the MAC contention window to better facilitate both concurrent transmission and QoS support. This greatly reduces the protocol complexity of the network system compared with other existing schemes. Further, we adopt a realistic directional antenna model to evaluate our proposed algorithm for the backhaul network in 73 GHz band. The simulation results demonstrate the superior performance of our algorithm against other existing schemes.

Having solved the communication system problem, we also notice another important problem. Nowadays, it is common for a computer system (e.g., desktop, laptop, tablet, mobile phone) to have a single user and never be shared. Also computer systems’ users are diverse and have rather different profiles when using a computer. For example, some people just use a browser for their computing needs, while others use computers as part of their work and run specific applications at specific times, which might include personal and task-oriented software. For example, consider Bob, 72, living in a retirement community in Florida. His desktop usage pattern is rather different than that of Alice, 25, working for a startup in San Francisco. Bob uses a browser to read the news and e-mails, and to check for events in his community. He also uses Skype to talk to his son in Illinois. Alice has MS Office installed on her laptop and regularly uses MS Word to write weekly reports for her manager. She also uses Chrome browser and regularly accesses sites hosted not only in the United States, but also in China regularly. She works from home Mondays and Wednesdays, and on these days, VPN and VNC client are running for connection with the company server. While Bob never accesses his computer past 8 p.m., Alice is usually active on her computer up to midnight.
The patterns of usage for Bob and Alice suggest that computer systems behave as if they possess a personalized “microbiota”. In biology, the microbiota is the large set of microbes that share our body with our human cells and outnumber human cells by a ratio of 10:1. The large majority of them are benign and influence, for better or worse, critical functions for our physiology, such as digestion, allergic reactions, and propensity to diseases, such as cancer or Alzheimer’s. Each human’s microbiota is rather unique, providing a signature for the individual, a signature which has been proposed to be applied even in criminal forensics [14]. The microbiota’s constitution depends on the human’s diet, lifestyle and geographic location. In spite of that, microbiota composition form clusters depending on people’s similarity for example, people living together, having the same lifestyle, living in the same geographic area, or having similar healthy conditions have similar microbiotas. We will provide details in the following chapters to give an insight to ways of properly approaching these problems.

Generative Adversarial Networks (GANs) [15] have gained widespread popularity and attentions in recent years. One might legitimately wonder why generative models are worth studying, especially generative models that are only capable of generating data rather than providing an estimate of the density function. GANs was first applied to the world of image studies. After all, when applied to images, such models seem to merely provide more images, and the world has no shortage of images. There are several reasons to study generative models.

First, training and sampling from generative models is an excellent test of our ability to represent and manipulate high-dimensional probability distributions. High-dimensional probability distributions are important objects in a wide variety of applied math and engineering domains.

Second, generative models can be incorporated into reinforcement learning in several ways. Reinforcement learning algorithms can be divided into two categories; model-based and model-free, with model-based algorithms being those that contain a generative model. Generative models of time-series data can be used to simulate possible futures. Such models could be used for planning and for reinforcement learning in a variety of ways.
model used for planning can learn a conditional distribution over future states of the world, given the current state of the world and hypothetical actions an agent might take as input. The agent can query the model with different potential actions and choose actions that the model predicts are likely to yield a desired state of the world. For a recent example of such a model, see [16], and for a recent example of the use of such a model for planning, see [17]. Another way that generative models might be used for reinforcement learning is to enable learning in an imaginary environment, where mistaken actions do not cause real damage to the agent. Generative models can also be used to guide exploration by keeping track of how often different states have been visited or different actions have been attempted previously. Generative models, and especially GANs, can also be used for inverse reinforcement learning.

Third, generative models can be trained with missing data and can provide predictions on inputs that are missing data. One particularly interesting case of missing data is semi-supervised learning, in which the labels for many or even most training examples are missing. Modern deep learning algorithms typically require extremely many labeled examples to be able to generalize well. Semi-supervised learning is one strategy for reducing the number of labels. The learning algorithm can improve its generalization by studying a large number of unlabeled examples which, which are usually easier to obtain. Generative models, and GANs in particular, are able to perform semi-supervised learning reasonably well.

Fourth, generative models, and GANs in particular, enable machine learning to work with multi-modal outputs. For many tasks, a single input may correspond to many different correct answers, each of which is acceptable. Some traditional means of training machine learning models, such as minimizing the mean squared error between a desired output and the model’s predicted output, are not able to train models that can produce multiple different correct answers [18].

Finally, many tasks intrinsically require realistic generation of samples from some distribution, see [19–22] for example works.
This dissertation is organized as follows: Chapter 2 describes our proposed practical distributed scheduling framework. Chapter 3 discusses our proposed framework for user profile anomaly detection based on dimension reduction and proposed bi-directional attention generative adversarial networks. Chapter 4 proposes an alternative approach using generative adversarial networks to the same problem as raised in Chapter 3. Chapter 5 finally concludes this dissertation summarizes the key points presented in this dissertation.
This chapter describes DMQ – a practical and efficient distributed scheduling for QoS-aware small cell mmwave mesh backhaul network.

2.1 Problem Formulation

Unlike existing communication systems using lower carrier frequencies (e.g., from 900 MHz to 5 GHz), mmWave communications suffer from high propagation loss. To combat severe channel attenuation, directional antennas are utilized at both the transmitter and receiver for high antenna gain. With the beamforming technique, the transmitter and the receiver are able to direct their beams towards each other for the directional communication \[23\]. The directional communication reduces the interference between links, and concurrent transmissions (spatial reuse) can be exploited to greatly improve network capacity. In a scenario where small cells are densely deployed, effective and efficient backhaul scheduling schemes need to be designed with the characteristics of mmWave communications taken into account.

There has been some related work on the transmission scheduling for mmWave communications. Since time division multiple access (TDMA) is adopted in ECMA-387 \[7\] and IEEE 802.15.3c \[8\], many existing works are based on TDMA \[24–29\]. One influential work is the Exclusive Region (ER) based scheduling which is introduced and derived in \[26\]. Qiao et al. \[28\] proposed a concurrent transmission scheduling considering QoS for flows. In such work, the set of concurrent flows are chosen in a greedy manner to maximize the overall system throughput, through which the number of flows successfully scheduled is maximized.

Although TDMA-based scheduling has been widely adopted in literature, one of its key disadvantage is that a centralized backhaul network controller is needed to coordinate scheduling process. This inevitably increases the system overhead, complicates the system design and limits the scalability of backhaul network. Furthermore, as energy efficiency emerges...
to be an important metric for next generation wireless networks [30], it also poses a challenge that the increased overhead and complexity would increase the energy cost.

In this dissertation, we develop a practical CSMA contention based scheduling scheme that is also QoS aware for the small cell backhaul network specifically in mmWave band. The proposed scheme operates in a completely distributed fashion and does not need centralized controller to coordinate flow scheduling. To the best of our knowledge, this is the first CSMA contention based scheduling protocol that prioritizes the MAC contention window to better facilitate both concurrent transmission and QoS support. This greatly reduces the protocol complexity of the network system compared with other existing schemes. Further, we adopt a realistic directional antenna model to evaluate our proposed algorithm for the backhaul network in 73 GHz band.

2.2 System Model

2.2.1 Network Model

The characteristic of strong line of sight (LOS) at low inter-site distances (ISDs) of 73GHz mmWave band generally requires the small cells be deployed densely. Therefore, we consider the scenario where small cells are densely deployed as illustrated in Fig. 2-1, where we illustrate a typical scenario of densely deployed small cells underlying the macrocell cellular network. In the small cells, mobile users are associated with the base stations (BSs), and the BSs are connected via backhaul links with the mesh topology. There are one or more BSs connected to the backbone network via the macrocell site, which are called gateways. Each BS in the network is equipped with an electronically steerable directional antenna, and can direct its beam towards other BSs for directional transmission. The backhaul network is in E-band, which provides high data rates. For the scheduling problem of such backhaul network for small cells, there are two challenges. First, concurrent transmissions need to be fully exploited to maximize the spatial reuse gain. Second, the scheduling scheme should provide the quality of service (QoS) guarantee for flows in the backhaul network. To ensure fairness, the scheduling
scheme should maximize the scheduled flows in the network with the QoS requirement of each flow satisfied.

Figure 2-1. The mesh backhaul network in the small cells densely deployed scenario.

2.2.2 Directional Antenna Model

With BSs fixed, since non-line-of-sight (NLOS) transmissions suffer from higher attenuation than line-of-sight (LOS) transmissions [31], we assume the directional LOS transmission between BSs can be achieved with the locations of BSs adjusted appropriately (e.g., on the roof). Suppose there are \( N \) flows requesting transmission, and each flow represents one backhaul link. We denote the distance between the transmitter \( s_i \) of flow \( i \) and the receiver \( r_j \) of flow \( j \) by \( d_{ij} \). We also denote the antenna gain of \( s_i \) in the direction of from \( s_i \) to \( r_j \) by \( G_t(i,j) \), and the antenna gain of \( r_i \) in the direction of from \( s_j \) to \( r_i \) by \( G_r(j,i) \). Then considering the path loss and signal dispersion over distance, the received power at the receiver \( r_i \) from \( s_i \) can be calculated as

\[
P_r(i,i) = k_0 G_t(i,i) G_r(i,i) d_{ii}^{-n} P_t, \tag{2–1}
\]
where $k_0$ is a constant coefficient and proportional to $(\frac{\lambda}{4\pi})^2$ ($\lambda$ denotes the wavelength), $n$ denotes the path loss exponent, and $P_t$ denotes the transmission power [26, 28]. Due to the half-duplex assumption, adjacent links cannot be scheduled for concurrent transmissions [32]. If flow $i$ and flow $j$ are not adjacent, we denote it by $i \propto j$. Then under concurrent transmissions, the received interference at $r_i$ from $s_j$ can be calculated as

$$P_i(j, i) = \rho k_0 G_t(j, i) G_r(j, i) d_{ji}^{-n} P_t.$$  \hspace{1cm} (2–2)

where we consider $\rho$ as the multi-user interference (MUI) factor related to the cross correlation of signals from different links. Thus, the received SINR at $r_i$ can be expressed as

$$\text{SINR}_i = \frac{k_0 G_t(i, i) G_r(i, i) d_{ii}^{-n} P_t}{N_0 W + \rho \sum_{j\propto i} k_0 G_t(j, i) G_r(j, i) d_{ji}^{-n} P_t},$$  \hspace{1cm} (2–3)

where $W$ is the bandwidth, and $N_0$ is the onesided power spectra density of white Gaussian noise [28]. Then according to Shannon’s channel capacity, the achievable data rate of flow $i$ can be estimated as

$$R_i = \eta W \log_2(1 + \frac{k_0 G_t(i, i) G_r(i, i) d_{ii}^{-n} P_t}{N_0 W + \rho \sum_{j\propto i} k_0 G_t(j, i) G_r(j, i) d_{ji}^{-n} P_t}),$$  \hspace{1cm} (2–4)

where $\eta \in (0, 1)$ describes the efficiency of the transceiver design.

For individual antenna, we adopt the reference model described in [33]. First, we define maximum gain of antenna as $G_0$, which can be determined by employing the formula of an ideal circular aperture antenna with uniform field distribution:

$$G_0 = (ka)^2$$  \hspace{1cm} (2–5)

$$ka \cdot \sin(\theta_{-3dB}/2) = 1.6162$$  \hspace{1cm} (2–6)

where $k$ and $a$ are the wavenumber and radius of the aperture.

Then, the main-lobe function $G$ is defined as

$$G(\theta, \phi) = G_0 \exp(-\alpha \theta^2).$$  \hspace{1cm} (2–7)
where $\theta$ is the main-lobe width and $\alpha$ is determined by the half-power beamwidth $\theta_{-3dB}$ as:

$$
\alpha = \frac{4 \cdot \ln 2}{\theta^2_{-3dB}}. \tag{2-8}
$$

In unit of dB, the main-lobe can be expressed as:

$$
G(\theta, \phi)[dB] = G_0[dB] - 3.01 \cdot \left(\frac{2\theta}{\theta_{-3dB}}\right)^2. \tag{2-9}
$$

Considering -20 dB from the maximum gain as a main-lobe, the main-lobe width $\theta_{ml}$ can be derived from (2-9) as:

$$
\theta_{ml} = 2.6 \cdot \theta_{-3dB}. \tag{2-10}
$$

Finally, the side-lobe power $G_{sl}$ can be modeled as:

$$
G_{sl} = \frac{4\pi - G_0 \int_0^{\theta_{ml}/2} \exp(-\alpha \theta^2) \cdot \sin \theta d\theta \int_0^{2\pi} d\phi}{\int_0^{\pi} \sin \theta d\theta \int_0^{2\pi} d\phi}. \tag{2-11}
$$

### 2.3 Problem Formulation

In this section, we formulate the optimal scheduling problem as a nonlinear integer programming problem.

We assume there is a QoS requirement that represents a minimum throughput requirement for each flow $i$, and denote it by $q_i$. We also define a term grouping, which represents a group of nodes that are able to schedule transmissions concurrently. We denote a schedule as $S$, and assume it has $K$ groupings. In each grouping, multiple links are scheduled for concurrent transmissions. For each flow $i$, we define a binary variable $a^k_i$ to indicate whether flow $i$ is scheduled in the $k$th grouping. If so, $a^k_i = 1$; otherwise, $a^k_i = 0$. We denote the number of time slots of the $k$th grouping by $\delta^k$.

Since there are different links in different groupings, we denote the transmission rate of flow $i$ in the $k$th grouping by $R^k_i$. Then we can obtain $R^k_i$ as

$$
R^k_i = \eta W \log_2 \left(1 + \frac{a^k_i k_0 G_t(i, i)G_r(i, i)d_{ii}^{-n} P_t}{N_0 W + \rho \sum_j a^k_j k_0 G_t(j, i)G_r(j, i)d_{ji}^{-n} P_t}\right). \tag{2-12}
$$
Then we can obtain the throughput of flow $i$ based on $S$ as

$$T_i = \frac{\sum_{k=1}^{K} \delta^k \cdot R_i^k \cdot t_{\text{slot}}}{t_{\text{bo}} + M \cdot t_{\text{slot}}},$$

(2–13)

where $t_{\text{bo}}$ is the time for back-off, and $t_{\text{slot}}$ is the time duration of each time slot on transmission. $M$ denotes the total number of time slot available. Then we define a binary variable $Q_i$ to indicate whether the QoS requirement of flow $i$ is satisfied in $S$. If so, $Q_i = 1$; otherwise, $Q_i = 0$. Given the throughput requirements of flows, with limited number of time slots assigned, the optimal schedule should accommodate as many flows as possible [28]. Therefore, the optimal scheduling problem $P$ can be formulated as follows.

$$(P) \quad \max \sum_{i=1}^{N} Q_i$$

s.t.

$$Q_i = \begin{cases} 
1, & \text{if } T_i \geq q_i, \quad \forall \ i \\
0, & \text{otherwise};
\end{cases}$$

(2–15)

$$\sum_{k=1}^{K} \delta^k \leq M;$$

(2–16)

$$a_i^k + a_j^k \leq 1, \quad \text{if flow } i \text{ and } j \text{ are adjacent}; \quad \forall \ i, j$$

(2–17)

It is obvious that this is a nonlinear integer programming problem, and is NP-hard. Constraint (2–15) indicates if the throughput of flow $i$ in the schedule is larger than or equal to its throughput requirement, $Q_i = 1$; otherwise, $Q_i = 0$. Constraint (2–16) indicates there are at most $M$ time slots. Constraint (2–17) indicates due to the half-duplex operation of BSs, adjacent links cannot be scheduled for concurrent transmissions since there is at most one connection for each node [32].
Although it is difficult to solve the problem $\mathbf{P}$ in polynomial time, we propose an efficient and practical distributed scheduling algorithm to achieve a sub-optimal solution instead in the next section.

### 2.4 Distributed Scheduling Algorithm

In this section, we propose a practical Distributed Maximum QoS-aware (DMQ) scheduling algorithm to solve the scheduling problem in (2–14).

#### 2.4.1 Neighborhood Detection

In order to effectively schedule flows between base stations, each station needs to keep track of how many other stations that can potentially cause interference to the station. One of the advantage of mmWave band is its strong capability for channel reuse. However, interference plays an important role in reducing such channel reuse capability. By detecting information of neighbors, we are able to reduce such interference. One straight-forward way to conduct neighborhood detection could be that every node periodically broadcasts beacon signal to its neighbors. If the other node is able to receive such beacon signal, then it knows it has one neighbor that could possibly in the interference range. However, since interference only occurs when nodes are transmitting, such active signaling mechanism can cause unnecessary overhead over time. The increased overhead will not only impact the performance of the network, but it also increases the energy consumption. We can eliminate such overhead by setting up an overhearing mechanism at each base station.

The neighborhood detection algorithm works as follows. At time slot $t$, each base station $b_i$ counts the number of distinct sources for flows not destined to $b_i$ by letting the NIC operate in promiscuous mode. Therefore, when a nearby base station $b_j (j \neq i)$ sends a packet, base station $b_i$ is able to detect the signal and effectively count $b_j$ as its neighbor. The detailed procedure is described in Algorithm 2.1.

**Algorithm 2.1. Neighborhood Detection**

*Each base station does the following:*  
*Output:* the number of interfering neighbors detected.  
1. Enable promiscuous mode;
2: Define and initialize a neighbor set $\mathbf{N}$;  
3: while received a packet $p$ do  
4:  $\text{src} = \text{source address of } p$;  
5:  $\text{dest} = \text{destination address of } p$;  
6:  if dest is not a broadcast address then  
7:  Define $b_{\text{src}}$ as the base station that sent packet $p$;  
8:  Add $b_{\text{src}}$ to neighbor set $\mathbf{N}$;  
9:  end if  
10: end while  
11: return size of $\mathbf{N}$;  

While signal in mmWave band is more concentrated and less likely that it will cause interference in large range, yet it is still not uncommon that interference does happen when several base stations are located in densely deployed small cells and, in the meantime, the directional antennas’ orientation is also set within the interfering sidelobe or mainlobe area. Thus, for a base station $b_i$, there could be simultaneously several neighbor base stations contending for the same channel. This will cause the degradation of network throughput if we don’t take measures to mitigate the problem. Our proposed DMQ is right designed for this task. DMQ scheduler is able to schedule flows according to a base station’s contention level, which is positively proportional to its number of neighbors.

2.4.2 Contention Window Prioritization

In order to facilitate both contention avoidance and QoS-aware scheduling between base stations, we define eight priority levels indexed from 0 to 7. For practical concern, DMQ algorithm exploits the Quality of Service (QoS) feature shipped with 802.11e. IEEE 802.11e standard is an amendment to IEEE 802.11 standard family that defines a set of QoS facilitating features, of which the most notable is the Enhanced Distributed Channel Access (EDCA) mechanism. EDCA provides levels of priority which are called Access Categories (ACs). By default, there are 8 distinct ACs indexed from 0 to 7. We follow such settings in order to make our proposed scheme adopt to 802.11e standard. Each AC has a pair of minimum contention window size ($\text{CWmin}$), maximum contention window size ($\text{CWmax}$). These parameters regulate the back-off behavior of current distributed coordination function (DCF). The larger index number represents higher priority level and vice-versa. Each priority
level has parameters of minimum contention window size (CWmin) and maximum contention window size (CWmax). The configuration of these parameters are shown in Table 2-1.

For a specific priority, the CWmin and CWmax regulate the back-off behavior of current contending base station. When contention occurs, similar to CSMA/CA mechanism, the scheduler will initialize a random back-off window of size within CWmin and CWmax. If contention still occurs after back-off, the scheduler will double the size of current back-off window until it reaches CWmax and will keep it at CWmax until reset. The back-off procedure continues. Generally, the average size of back-off window determines how contentious current base station is. The larger the size of back-off window, the longer the base station will wait to try to initiate next transmission, thus exhibiting less contentious behavior, and vice-versa.

Therefore, we utilize this fact to configure the CWmin and CWmax parameters with respect to their associated priority levels. For contention resolving purposes, we want the base stations operating in lower priority level to be less contentious. Thus we assign them with relatively larger CWmin and CWmax values. Vice-versa, the base stations with higher priority levels will be assigned with smaller CWmin and CWmax values. Furthermore, we also try to adjust CWmin and CWmax to make each priority level more distinct and disparate from one another; such that, for a given priority level $i$, the probability that the randomized back-off window size will overlap with that of priority level $j (j \neq i)$ will be reduced to minimum. Table 2-1 shows the exact configuration of these parameters, where we completely eliminate the possible back-off window size overlapping between different priority levels.

### 2.4.3 Coarse-to-Fine Window Mapping Algorithm

Based upon prioritized contention window configuration in Table 2-1, we propose a Coarse-to-Fine Window Mapping (CFWM) algorithm. Namely, CFWM is a two-phase process.

<table>
<thead>
<tr>
<th>Priority</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWmin</td>
<td>512</td>
<td>256</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>CWmax</td>
<td>1023</td>
<td>511</td>
<td>255</td>
<td>127</td>
<td>63</td>
<td>31</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 2-1. Contention Window Prioritization
In the coarse-phase, the algorithm addresses the contention issue while in the fine-phase, QoS requirements are accommodated.

2.4.3.1 Coarse phase

Let’s first define contention degree $d_c^i$, as the number of neighboring base stations a base station $b_i$ detects. The value of $d_c^i$ will be calculated by Algorithm 2.1 described in Section 2.4.1. CFWM works by assigning priority level to current schedule according to its base station $b_i$’s contention degree $d_c^i$. We call this coarse phase of CFWM, which is illustrated in Algorithm 2.2.

Algorithm 2.2. Priority Level Computation (Coarse)

**Input:** contention degree $d_c^i$ for base station $b_i$.

**Output:** the corresponding priority level.

1: Define $P_i$ as the priority level to be assigned for base station $b_i$;
2: if $d_c^i > 7$ then
3: $P_i = 0$;
4: else
5: $P_i = 7 - d_c^i$;
6: end if
7: return $P_i$;

The intuition behind Algorithm 2.2 is that the scheduler always tries to assign higher priority level to base station $b_i$ that has smaller value of contention degree $d_c^i$. In such case, since the base station has less number of neighbors contending with it (smaller value of contention degree), it is more likely that this base station is able to fulfill the QoS requirement and increase the overall network throughput of the system. Thus, CFWM will assign it with a higher priority level. On the other hand, if base station $b_i$ has a larger value of contention degree, the CFWM will assign it with a lower priority level which tends to curb the sending rate of base station $b_i$. In practice, the number of contention neighbors a base station potentially has usually would not exceed 7, thus we believe the linear mapping between contention degree and priority level (line 2–5 in Algorithm 2.2) is reasonable. For cases where contention degree does exceed 7, CFWM just assigns priority level 0, the lowest, to the corresponding base stations.
In essence, each priority level determines the contention window adjustment strategy. Since the lower the priority level is, the larger CWmin/CWmax will be set, which leads to a less contentious base station. It is also true vise-versa. Our rationale behind this mechanism is that we always want the least contending base station to transmit first, because the less contending a base station is, the higher throughput the base station can potentially create. As will be seen in Section 2.5, our experimental results also support this rationale.

2.4.3.2 Fine phase

So far, the priority level assignment procedure described in Algorithm 2.2 only considers the contention between base stations. The more neighbors a base station $b_i$ has, the more contending $b_i$ becomes, the lower priority level $b_i$ will be assigned. Another important issue that we need to address is QoS-awareness of the DMQ scheduler. On top of the scheduling scheme in Algorithm 2.2, we equip DMQ with QoS scheduling capability which is described in Algorithm 2.3. We call this fine phase of CFWM. 

Algorithm 2.3. Contention Window Adjustment (Fine)

**Input:**
- $P_i$ – current priority level for base station $b_i$;
- $d_q^f$ – QoS degree for base station $b_i$.

**Output:**
- $CW_{min}$ – the adjusted CWmin value.
- $CW_{max}$ – the adjusted CWmax value.

1: Retrieve CWmin and CWmax values with respect to $P_i$;
2: Let $f(t) = \frac{2}{1 + e^{-\alpha t}} - 1$, where $\alpha$ is a constant;
3: $CW_{max} = CW_{min} + f(d_q^f) \cdot (CW_{max} - CW_{min})$;
4: $CW_{min} = \max(CW_{max} - \beta, CW_{min})$, where $\beta$ is a positive integer;
5: return $CW_{min}$ and $CW_{max}$.

For each flow $f$, define QoS degree $d_q^f$ as a positive value that indicates the QoS priority for the flow. The higher the QoS priority is, the larger $d_q^f$ is assigned. In Algorithm 2.3, CFWM tries to make a finer contention window parameter adjustment based on QoS degree $d_q^f$ for current flow $f$ on base station $b_i$. According to our QoS-aware schedule policy, the higher QoS priority a flow has, the sooner it needs to be scheduled in order to achieve optimal throughput over the network. CFWM achieves this purpose by adjusting the CWmin and CWmax value.
according to the QoS priority inside current priority level $P_i$. Recall in Section 2.4.2, we described that CWmin and CWmax determine the range within which the size of contention window could be. Although not guaranteed, statistically, when CWmin or CWmax increases, the back-off time for current DCF will become longer since the probability to randomize a larger back-off slot number becomes higher. Therefore, when a flow with higher QoS priority needs to be scheduled, Algorithm 2.3 will return a decreased CWmax and CWmin value (line 3 and line 4 of Algorithm 2.3) which essentially leads to a shorter back-off time. This will make the flow be able to have much larger probability to be scheduled earlier than its contending counterparts if there are any.

Algorithm 2.2 and Algorithm 2.3 altogether constitute the cornerstone of our proposed CFWM algorithm. They work cooperatively to schedule flows according to both their contention degree and QoS degree. When a base station $b$, has a flow to send, CFWM will first retrieve appropriate priority level by referring to Algorithm 2.2 with contention degree information. Afterwards, CFWM will continue to run Algorithm 2.3 to set proper CWmin and CWmax values for scheduler’s back-off window inside current priority level. Results returned from Algorithm 2.2 only indicates which priority level current flow should be associated with, while Algorithm 2.3 further deals with the exact values of CWmin and CWmax inside that priority level. Thus Alogrithm 2.2 is called coarse phase while Algorithm 2.3 is called fine phase. That is how CFWM deals with both contention issue and QoS-awareness in a unified framework.

2.5 Simulation Results

In this section, we implemented our proposed DMQ algorithm and compared the performance with existing schemes.

2.5.1 Development of Simulator

For performance evaluation of DMQ, we consider a backhaul network consisted of 25 base stations (nodes) evenly distributed over a 500 meters $\times$ 500 meters grid area, as illustrated in Figure 2-2. Each base station is equipped with directional antennas as well as quasi
omni-directional antennas. For each node, we implemented the widely adopted directional antenna model as described in [33]. The beamwidth of the directional antennas is set to 30 degrees and the channel is set to work on 73GHz band. On top of directional antenna model, our proposed DMQ scheduling algorithm is implemented. The simulation system is developed under NS-3 [34].

For each base station, it is able to communicate with its 8 adjacent neighbors in one-hop distance. We note that for base stations that are out of one-hop distance, an appropriate routing protocol is needed to route packets through multiple hops. Since routing is not our focus in this work, we believe many existing wireless routing protocols should work with our experiment. For simplicity, we just use Optimized Link State Routing (OLSR) protocol to achieve such purpose.

For performance comparison, we also implemented the optimal Distributed Opportunistic Scheduling (DOS) scheme proposed in [13] and a default 802.11 random access scheme as a baseline. To evaluate our proposed scheme, the following metrics are considered:

- **System throughput**: Needless to say, system throughput is a key performance metric we primarily concern about. By throughput, we mean the overall throughput on the whole backhaul network. We will study the effects of number of flows, effects of number of packets and effects of antenna beamwidth on throughput performance.

- **Packet loss ratio**: Since we also consider QoS as an essential part of our scheduling, packet loss ratio is yet another important indicator that measures the reliability performance of the scheduling system from a packet level perspective. We will study the effects of number of flows on packet loss ratio.

### 2.5.2 Performance Evaluation

In this subsection, we design experiments to study the scheduling algorithm performance under different number of flows, under different number of packets in each flow and under different antenna beamwidth. In order to focus on scheduling itself rather than the effect of routing, we use the term “flow” to only refer to the case where the flow is originated at one base station and travels to another in one hop. If there are more than one hop along the path of the flow, e.g. $k(k > 1)$ hops, we regard this as $k$ flows. To reduce random error for each
Figure 2-2. Topology of base stations used in simulation.

experiment, we repeat the same experiment for 10 times and the average results are calculated and taken.

2.5.2.1 Effects of number of flows on throughput performance

First, we set that each flow is consisted of 10 packets and each packet is of size 1000 bytes. Then, we add flows to the system. The number of flows is varied in range of 10 to 100. Once the number of flows is determined, these flows are generated and added randomly among the base stations. For each base station, one packet is scheduled to be generated and sent from transport layer in every 30 milliseconds, i.e., the sending interval. On the MAC layer, we configured each base station and installed scheduling scheme using, respectively, DMQ, DOS as well as default 802.11 random access. We recorded the elapsed time between moment when
first packet of the whole network was sent at transport layer and moment when last packet of the whole network was successfully received at one base station. Then we calculated the overall system throughput for each scheme. We recorded throughput performance of each scheme for different number of flows. Each of the experiment was repeated 10 times and the average results were taken. Finally, we plotted the comparison results shown in Figure 2-3.

![Image](image.png)

**Figure 2-3.** Number of flows on throughput performance.

We have the following observations:

- Our proposed DMQ achieves the best result in system throughput performance among all of the three schemes. It achieves an average of 25% throughput gain over DOS and 40% over 802.11. Thanks to the coarse-to-fine hierarchical structure of DMQ, the scheduler is able to make best use of the available contention window values and assigns them to flows according to both their contention and QoS priorities. Thus the contention between different flows with different priorities can be reduced to minimum.

- Our proposed DMQ is better suited for practical environment. From Figure 2-3, we see that DOS does not achieve the best performance although it is theoretically proved to be optimal. This is due to the fact that DOS relies on highly accurate past channel estimation to determine current scheduling scheme. However, in practical environment, it
is difficult to achieve such an accurate channel estimation in real time. This makes DMQ perform much better than DOS.

2.5.2.2 Effects of number of packets in each flow on throughput performance

In this experiment, we randomly add 50 flows in the system. Then, we vary the number of packets for each flow from 10 to 100. Other setups remain the same as described in Section 2.5.2.1. As previously, we recorded the elapsed time between moment when first packet of the whole network was sent at transport layer and moment when last packet of the whole network was successfully received at one base station. The performance comparison is shown in Figure 2-4.

![Figure 2-4. Number of packets in each flow on throughput performance.](image)

It clearly shows that:

- Our DMQ outperforms both DOS and default 802.11 scheduling schemes in terms of throughput performance. DMQ can achieve more than 20% gain over DOS and 40% gain over 802.11 as the number of packets increases.
The general trend is that when the number of packets in each flow increases, the overall system throughput also increases.

As the number of packets increases, the increasing trend of system throughput tends to slow down at the end. This is because the system is more and more approaching its capacity as more and more packets are injected into the system.

2.5.2.3 Effects of antenna beamwidth on throughput performance

One of the major differences between omni-directional antenna and directional antenna is the beamwidth. In directional antenna patterns, the beamwidth is the angle between half-power (-3dB) points and the main lobe. Therefore, beamwidth is an important factor that affects the performance of directional antennas. In this experiment, we study the effects of antenna beamwidth on system throughput. We set the number of flows to 50 and randomly assigns them to base station pairs in the system. In each flow, we set the number of packets to 50. Then, we vary the beamwidth from 5° to 50°. As previously, we recorded the elapsed time between moment when first packet of the whole network was sent at transport layer and moment when last packet of the whole network was successfully received at one base station. We collect the experiment data and compare the result in Figure 2-5.

We observe that:

- Our proposed DMQ achieves the best throughput performance among all the three schemes compared. It can achieve a maximum of 30% gain over DOS and even triple the performance of 802.11.

- The system throughput decreases as the beamwidth increases. This is because smaller beamwidth means a narrower antenna pattern between half-power and main lobe, which would consequently make the wireless signal more concentrated. As a result, the interference between base stations will be reduced, making the whole system less contending. Thus the throughput can be achieved a lot higher.

- When beamwidth keeps increasing, the contention between base stations also increases as more and more base stations become able to detect the signals from neighboring base stations. This leads to the trigger of back-off mechanism, thus reducing the system throughput.
2.5.2.4 Effects of number of flows on packet loss ratio

In this experiment, we focus on the relationship between number of flows in the system and its corresponding packet loss ratio. The experiment setup and methodology are similar to the those described in Section 2.5.2.1. However, instead of recording throughput, here we record the total number of packets that are successfully received by all receiving base stations. Using such information, we are able to calculate the packet loss ratio (PLR) as:

\[
PLR = 1 - \frac{\text{Total number of packets received}}{\text{Total number of packets sent in the network}}.
\]  

(2-18)

Again, the experiment was repeated 10 times and the average results were taken. The results of comparison is shown in Figure 2-6.

We have the following observations:
Our proposed DMQ achieves the least packet loss ratio which indicates the best performance among the three. It almost can achieve close-to-zero packet loss ratio when the number of flows is small.

Thanks to CFWM algorithm, DMQ is able to detect and avoid the contention as much as possible, and only loses half of packets DOS has lost and a quarter of packets conventional 802.11 has lost.

The packet loss ratio of all three schemes tends to increase as the number of flows increases. This is because when number of flows increases, the probability of contention between different flows is also increasing, which leads to the increase of packet loss ratio. Although increasing, the proposed DMQ always keeps packet loss ratio much lower than others.

2.6 Chapter Summary

In this chapter, we have considered the problem of backhaul scheduling to maximize the system throughput while having flows’ QoS requirements satisfied in the mmWave backhaul network. The formulated problem is NP-hard, thus we proposed the DMQ scheduling algorithm, where it is designed to achieve the spatial reuse while reducing interference and
channel contention between neighbors in a coarse-to-fine two-phase fashion. The QoS aware priority is exploited in the algorithm to provide better QoS guarantees for flows. We conducted extensive experiments and results have shown that our proposed algorithm is able to achieve much higher system throughput and much lower packet loss ratio than other existing schemes under different criteria.
CHAPTER 3
BI-DIRECTIONAL ATTENTION GENERATIVE ADVERSARIAL NETWORKS FOR USER PROFILE IDENTIFICATION AND ANOMALY DETECTION

This chapter describes attention GAN framework we have proposed to achieve the goal of identification and anomaly detection of user profiles.

3.1 Problem Formulation

Protecting end-user devices and computer system infrastructure is an ongoing problem, and today’s solutions have not evolved significantly over time. Specifically, the industry still mostly relies on antivirus-based solutions, which are almost exclusively using signature-based detection of threats [35, 36]. Although showing good accuracy for known malware, these solutions cannot detect zero-day malware and have difficulties identifying polymorphic and metamorphic attacks, with a practical detection rate of 25%-50% [37]. Behavioral-based solutions have also been adopted [38, 39], focusing on identifying behavioral properties of the device, such as peculiar sequences of system calls, and using this information to distinguish patterns that characterize malware. Research has shown, however, that behavioral-based detectors are not always effective, presenting high false-positive rates [40], because of the increasing complexity and diversity of software.

To make matters worse, malware is evolving, and current security solutions for computer infrastructure and situation awareness are not coping with the increasing level of sophistication of attacks. For example, organizations are now targets of advanced persistent threat (APTs) attacks, highly planned and orchestrated infiltrations combining different types of exploits and malware [41], which can be caused by insiders and outsiders. APT malware is challenging for current solutions because it blends in with approved corporate software and traffic, and because it acts slowly, based on triggers.

Today’s security solutions have two main shortcomings. First, they usually come in an "one-size-fits-all" format, as they are designed for "any" computer system, and make decisions about computer system events while ignoring the unique way different people use computers. Second, these solutions usually operate at one specific layer of abstraction (e.g.,
the application, the network or the OS syscall layer), and can miss correlated and insightful observation from other layers.

Nowadays, it is common for a computer system (e.g., desktop, laptop, tablet, mobile phone) to have a single user and never be shared. Also computer systems’ users are diverse and have rather different patterns when using a computer. The patterns of usage behave as if they possess a personalized “profile”. Each person’s profile is rather unique, providing a signature for the individual, a signature which has been proposed to be applied even in criminal forensics [14]. Also, the profile composition form clusters depending on people’s behavior similarity. The problem largely falls into the realm of user profile anomaly detection and identification.

Anomaly detection is usually a well-studied problem. Anomaly detection is the task of identifying test data not fitting the normal data distribution seen during training. Approaches for anomaly detection exist in various domains, ranging from video analysis [42] to remote sensing [43]. They typically either use an explicit representation of the distribution of normal data in a feature space, and determine outliers based on the local density at the observations' position in the feature space. Carrera et al. [44] utilized convolutional sparse models to learn a dictionary of filters to detect anomalous regions in texture images. Erfani et al. [45] proposed a hybrid model for unsupervised anomaly detection that uses a one-class support vector machine (SVM). The SVM was trained from features that were learned by a deep belief network (DBN). The experiments in the aforementioned works were performed on real-life-datasets comprising 1D inputs, synthetic data or texture images, which have lower dimensionality or different data characteristics compared to medical images. An investigation of anomaly detection research papers can be found in [46]. In clinical optical coherence tomography (OCT) scan analysis, Venhuizen et al. [47] used bag-of-word features as a basis for supervised random forest classifier training to distinguish diseased patients from healthy subjects. Schlegl et al. [48] utilized convolutional neural networks to segment retinal fluid regions in OCT data via weakly supervised learning based on semantic descriptions of
pathology-location pairs extracted from medical reports. In contrast to our approach, both works used some form of supervision for classifier training. Seeböck et al. [49] identified anomalous regions in OCT images through unsupervised learning on healthy examples, using a convolutional autoencoder and a one-class SVM, and explored different classes of anomalies. In contrast to this work, the SVM in [49] involved the need to choose a hyper-parameter that defined the amount of training points covered by the estimated healthy region.

In this chapter, we propose a user profile identification and anomaly detection framework. The proposed framework consists of a data pre-processing module, i.e., feature building module, a dimension reduction module and a novel bi-directional attention GAN (AttenGAN) module. To the best of our knowledge, our proposed AttenGAN scheme is the first work that integrates attention LSTM into GAN framework and successfully applies GAN to user profile identification and anomaly detection tasks. The following sections describe our proposed framework in detail.

### 3.2 Building Feature Vectors

User data are complex. Ideally, we want to collect all available data from the system by our designed extractor [50] and let the collected data be analyzed. However, this is not usually possible due to several concerns:

- **Privacy** – Some sensitive personal user data are illegal to collect.
- **Performance** – The more data need to be collected, the higher performance is required for end-user systems to run the extractor without causing significant performance drain.
- **Storage overhead** – more data mean more storage and more network bandwidth will be required for transmitting these data.

In our system, we plan to collect information at the application, OS and network layers and associate all this information with the concept of process, the live entity of any computer system. In summary, we want to know which processes a user’s computer system runs, which network connections it makes, which files it accesses, whether or not it interacts with a user, and how it consumes memory.
Thus, the data we actually collect per process in our current implementation comprise of logs from file systems, network activity and process activity. File system data are composed of Process ID, File Path, and Access Time. Network activity data are composed of Process ID, IP Version, Sender IP Address, Sender Port, Receiver IP Address, Receiver Port, Connection Timestamp. Process activity data are composed of Parent ID, Process ID, Process Path, Timestamp. Since user activities will also depend on the time of the day and will vary on weekdays and weekends, that is the reason why timestamps are also collected. For example, Alice strives for a good work-life balance and does not work on weekends. Bob uses Skype to talk to his son only on Sunday afternoons.

Now that we have raw data from collected and they are logged in text format. Fig. 3-1 shows a sample log from file system data; Fig. 3-2 shows a sample log from network data, and Fig. 3-3 shows a sample log from process data.

It is clear that the information we collected is all text based. These logs cannot be properly further processed without being converted to numeric-based format. The first line of the log is always the date/time information indicating when the logs were created. The following lines are the detailed logs for each type of file system, process and network activities for each user. The technique we implement is to tokenize all these text-based information to numbers in order to create feature vectors in the first place. We also need to specify a resolution which indicates every how much time we tokenize one feature vector. The resolution for this study is set to one minute, that is, each feature vectors we used for training/testing is a collection of feature representations in one minute interval. Algorithm 3.1 shows how we build the basic text-to-number hashing for these data and the returned hash_bag is further used to create rows of feature vectors in Algorithm 3.2.

Algorithm 3.1. Hashing text to numbers

1. function Hashing(table)
2.     bag = set()                      \(\triangleright\) initialize empty set
3.     for each k \(\in\) table do
4.         for v \(\in\) table[v] do
5.             bag.add(v)

41
Figure 3-1. File system sample log

Figure 3-2. Network data sample log
Figure 3-3. Process data sample log

6:       end for
7:       end for
8:       idx = 1
9:       has_bag = dict()  ◄ initialize empty dict
10:      for each v ∈ bag do
11:         hash_bag[v] = idx
12:         idx += 1
13:      end for
14:      return hash_bag  ◄ text-to-number mapping
15:      end function

Algorithm 3.2. Hashing text to numbers

1:       function CREATE_ROW(table, key, hash_bag)  ◄ create feature vector rows
2:       v_len = size of (hash_bag)
3:       vec = zeros(v_len+1)  ◄ allocate zeros of count v_len+1. One more zero for unknown type
4:       if name ∈ hash_bag then
5:          vec[hash_bag[name]] = 1
6:       else
7:          vec[0] += 1
8:       end if
9:       return vec
10:      end function

The feature vectors we build are a concatenation of File system data, network data and process data that are hashed into number format. Each position of the element in the feature
vector row corresponds to one feature that exists in the dataset. Each feature vector is a high-dimension binary vector that tries to catch all the existing feature. However, we are aware that there may be cases where the feature exists in the dataset we currently have (e.g., training data) but does not exist in future data (e.g., testing data). Therefore, in each feature vector created, the first element is reserved to represent that ‘unknown’ token, and the rest each element is a binary value either 0 or 1, where 0 means the corresponding token doesn’t exist and 1 means it does. One sample feature vector should look like: [ 0., 0., 1., ..., 1., 0., 1.].

Finally, once we are able to create feature vectors, we are able to construct the feature vector matrix by concatenating each feature vectors together. The matrix will be the final input layer for deep learning framework. This process is shown in Algorithm 3.3.

**Algorithm 3.3. Create data matrix as input for machine learning**

1: function CreateDataMatrix $\triangleright$ data matrix for deep learning
2: lower = minimum timestamp of collected data
3: upper = maximum timestamp of collected data
4: for each $k \in \text{range}(lower:upper)$ do $\triangleright$ loop from range lower to upper
5: row = concatenation of feature vector for file system, process and network
6: data_matrix.append(row)
7: end for
8: return data_matrix
9: end function

### 3.3 Dimension Reduction

Once we have obtained the data matrix per Algorithm 3.3, it becomes natural to feed it to the machine learning system to train on the dataset. However, this is not often possible because the dataset is too large. For a simple instance, dataset we have for a typical user can accumulate to take up more than 2GB+ of disk space. If these data were directly fed into the training system, the memory consumption could easily exceed 100GB+ and the training time could increase indefinitely, which takes forever to finish training. Thus, we seek effective dimension reduction techniques to increase system efficiency while preserving the identification accuracy.
3.3.1 Literature Review

In the last few decades, a number of methods have been proposed for feature extraction and dimension reduction, including both well-known classical techniques and new approaches. These techniques aim to boost the general data analysis procedures by improving the characterization of features (efficacy) and/or relieving computational complexity (efficiency). For instance, features containing adequate information usually lead to higher identification accuracy, in many cases, this can be done along with a reduction in the number of features (feature dimensionality), which in turn increases the overall efficiency.

As a kind of common rule, superficially high-dimensional and complex phenomena can actually be dominated by a small amount of simple variables in most situations. Dimensionality reduction is an old and young, dynamic research topic [51, 52]. It is looking for a projection method that maps the data from high feature space to low feature space.

Methods focusing on feature representation include widely known classical techniques and, on the other hand, more modern approaches. Among the classical methods we can find principal component analysis (PCA) [53], which projects the original data onto its principal directions with the maximal variance, and does not consider any data relation. Linear Discriminant Analysis (LDA) [54] is a supervised method to find a linear subspace, which is optimal for discriminating data from different classes. Marginal Fisher Analysis (MFA) extends LDA by characterizing the intraclass compactness and interclass separability [55]. These two methods use class label information as a weak data relation to seek a low-dimensional separating subspace. Independent component analysis (ICA) [56], or maximum noise fraction (MNF) [57]. These techniques transform the data by means of a projection, with relation to distribution of variance, statistical independence, and noise ratio, respectively.

The following is a very quick introduction of four classical dimensionality reduction methods [51].
**PCA:** Principal component analysis is a very popular linear technique for dimensionality reduction. Given a dataset on R^n, PCA aims to find a linear subspace of dimension d lower than n which attempts to maintain most of the variability of the data.

**LDA:** Linear discriminant analysis is another popular linear dimensionality reduction method. The basic idea is to ensure the samples after projection to have maximum-between-cluster-distance and minimum-in-cluster-distance in the new subspace.

**LLE:** Locally linear embedding is a nonlinear approach to reduce dimensionality by computing low-dimensional, neighborhood preserving embedding of high-dimensional data. A dataset of dimensionality n, which is assumed to lie on or near a smooth nonlinear manifold of dimensionality d<n, is mapped into a single global coordinate system of lower dimensionality d. The global nonlinear structure is recovered by locally linear fits.

**Isomap:** Isomap is a nonlinear generalization of classical multidimensional scaling. The main idea is to perform multidimensional scaling, not in the input space, but in the geodesic space of the nonlinear data manifold. The geodesic distance represents the shortest paths along the curved surface of the manifold measured as if the surface were flat. This can be approximated by a sequence of short steps between neighboring sample points. Although these approaches were introduced quite a few years ago, they are still employed in various circumstances, and it is worth to highlight them.

In recent years, artificial neural network is a mathematical and computational model composed of a large number of neurons that can simulate the structural and functional characteristics of biological neural network. It is a self-adaptive system, which changes the internal structure according to the external input, and is commonly used to model the complex relationship between input and output. The development of artificial neural network is a tortuous road. Perceptron is the starting point of modern neural computation, and the proof of perceptron convergence theorem in 1962 triggered the first climax of research for neural network represented by perceptron. In 1965, Minsky and Papert [58, 59] pointed out the defects of perceptron and took a pessimistic view on the research of neural network, which
made neural network study from rise into stagnation. By the early 1980s, related work by Hopfield et al. [60] showed the potential of neural network which made neural network study from stagnation to boom. In the mid-1990s, with the advent of the support vector machine (SVM) [61], researchers realized some limitations of artificial neural network and the research for neural network fell into low tide period again.

Autoencoder, known as auto-association before, is a three-layered neural network and was studied by a number of researchers in 1990s. Bourlard and Kamp [62] discussed the relationship between auto-association by multi-layer perceptrons and singular value decomposition in 1988. They showed that for auto-association with linear output units, the optimal weight values could be derived by purely linear techniques relying on singular value decomposition and low rank matrix approximation. In 1991, Kramer [63] introduced how to conduct nonlinear principal component analysis using auto-associative neural networks with three hidden layers. Due to the difficulty in training, deep network with multi-layer stacked auto-encoders did not exert its superior strength for a long time. Some researchers committed themselves to investigating several fundamental issues of neural networks with one or two hidden layers [64–73].

Until 2006, Geoffrey Hinton [74] solved this problem in Science Magazine to a large extent and broke the stalemate that neural network was in low tide. The method was to do layer-wise pre-training [75] to multi-layer auto-encoders using RBM [76]. That becomes a very hot topic in recent years—deep learning. From then on, deep learning sweeps across the industry and the academia like a wave. Recent research results have also demonstrated that deep learning indeed achieves state-of-the-art performances among various areas [77–79]. Particularly, deep learning has already been successfully applied to the industry in many areas.

### 3.3.2 Autoencoders

The autoencoder algorithm belongs to a special family of dimensionality reduction methods implemented using artificial neural networks.
Suppose the original input $\mathbf{x}$ belongs to $n$-dimensional space and the new representation $\mathbf{y}$ belongs to $m$-dimensional space, an auto-encoder is a special three-layered neural network in which we set the output $h_{W,b}(\mathbf{x}) = (x_1, x_2, \ldots, x_n)^T$ equal to the input $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$. $J$ is the reconstruction error. It is an unsupervised learning algorithm and uses back propagation algorithm for training $h_{W,b}(\mathbf{x}) = g(f(\mathbf{x})) \approx \mathbf{x} J(W, b; \mathbf{x}, \mathbf{y}) = \frac{1}{2} ||h_{W,b}(\mathbf{x}) - \mathbf{y}||^2$.

Fig. 3-4 and Fig. 3-5 show the structure and the visualization description of an auto-encoder respectively. As shown in Fig. 3-5, from the first layer to the second layer amounts to an encoder $f$ and from the second layer to the third layer amounts to a decoder $g$. We then minimize the reconstruction error $J$ by adjusting parameters in the encoder and the decoder to get the code.

Figure 3-4. Structure of a typical autoencoder

Autoencoder can be seen as a way to transform representation. When restricting the number of hidden layer nodes $m$ greater than the number of original input nodes $n$ and adding
sparsity constraint, the result will be similar to sparse coding. When restricting the number of hidden layer nodes $m$ less than the number of original input nodes $n$, we can get a compressed representation of the input, which actually achieves desired dimensionality reduction effect.

In the process of dimensionality reduction, discarding some dimensions inevitably leads to the loss of information. Thus the primary problem to be solved is to keep the main and important characteristics of the original data as much as possible. So the process of dimensionality reduction is closely related to original data characteristics.

Intrinsic dimensionality, as an important intrinsic characteristic of high-dimensional data, can reflect the essential dimension of data well. Sample data in high-dimensional space generally cannot diffuse in the whole space; they actually lie in a low-dimensional manifold embedded in high-dimensional space and the dimensionality of the manifold is the intrinsic dimensionality of the data.

### 3.3.3 Stacked Autoencoders

Although there are many methodologies, it poses a challenge to find one that will work best for our dataset. The volume of the dataset suggests that we focus on a particular approach related to a new and really promising field, the deep learning (DL) framework, in particular with the study of stacked autoencoders (SAEs) [74, 77].

Based on neural network architectures, SAEs are able to reduce feature dimensionality to few elements contained in the deep layers of those networks. In SAEs, an input feature vector is introduced in the network by the first layer (or input layer), with as many nodes as original features in the feature vector. Then, the feature information travels the network through
subsequent layers with reduced number of nodes or units, to finally achieve a reconstructed feature vector at the output matching the original one. Therefore, SAEs can be employed effectively for feature extraction, where the abstraction level achieved in deep layers leads to representative reduced features. In that sense, the powerful capabilities from machine learning can be exploited to perform dimension reduction in such context that seems promising and needs proper investigation.

Stacked autoencoders is constructed by stacking a sequence of single-layer AEs layer by layer [80]. Fig. 3-6 illustrates an instance of an SAE with 5 layers that consists of 4 single-layer autoencoders. The single-layer autoencoder maps the input variables into the first hidden vector. After training the first single-layer autoencoder, the reconstruction layer of the first single layer autoencoder is removed, and the hidden layer is reserved as the input layer of the second single-layer autoencoder. Generally speaking, the input layer of the subsequent AE is the hidden layer of the previous AE. Each layer is trained using the same gradient descent algorithm as a single-layer AE and feeds the hidden vector into the subsequent AE. It is

![Illustration of a stacked autoencoder framework](image)

Figure 3-6. Illustration of a stacked autoencoder framework
noteworthy that the weights and bias of the reconstruction layer after finishing training each single-layer AE is cast away. Typically, the number of input variables can vary; then, the size of hidden layer can be tuned accordingly. Depth plays an important role in SAE because it determines qualities like invariance and abstraction of the extracted feature. The depth needs to be carefully chosen to achieve the best performance in a specific application [81].

3.3.4 Design and Implementation

As we mentioned in Section 3.2, A typical user’s data will finally reduced to a high-dimensional feature vector that can exceed a dimension of greater than 111 thousand. The dimension reduction module is designed based on a deep stacked autoencoder which reduces dimension from 111k+ to 1024. The structure is shown in Fig. 3-7. Layer 1 is the input layer which accepts the high dimensional input up to dimension of 111k+. Layer 7 is the decoder layer, i.e., reconstruction layer that generates the reconstructed output which can then be compared with input to get reconstruction error (Fig. 3-5). Layer 1 through Layer 4 are encoder layers whereas Layer 5 through Layer 7 are decoder layers. Layer 4 is our focus here which will, after training, output the dimension reduced results that finally are used to feed into consequent machine learning system.

The system works by first getting the original sample as input (e.g., dim=111569), then train the 7-layer deep stacked autoencoder. After training, we feed into the system input samples, then the output from the Encoder block is the dimension-reduced features. Those features are only with dimension 1024 rather than original 111569, which is greatly reduced and made more efficient.

Algorithm 3.4 shows the training algorithm of this dimension reduction module. Deep learning toolkit Keras [82] is used for this implementation.

**Algorithm 3.4. Training of dimension reduction module**

1: **procedure** TRAINDIMENSIONREDUCTION(input_dim, output_dim) \(\triangleright\) output_dim specifies the desired dimension
2: \[\text{layer}_1 = \text{CreateInputLayer}(\text{input}\_\text{dim})\]
3: \[\text{layer}_2 = \text{Dense}(4096, \text{activation}='\text{relu}')(\text{layer}_1) \triangleright\] dense layer on top of layer_1 with 4096 dimensions and activation function RELU

51
Figure 3-7. Implementation of a stacked autoencoder for our dataset

4: \[ \text{layer}_3 = \text{Dense}(2048, \text{activation}='relu')(\text{layer}_2) \]
5: \[ \text{layer}_4 = \text{Dense}(\text{encoding_dim}, \text{activation}='relu')(\text{layer}_3) \]  \( \triangleright \) ‘encoding_dim’ is the reduced dimension
6: \[ \text{layer}_5 = \text{Dense}(2048, \text{activation}='relu')(\text{layer}_4) \]
7: \[ \text{layer}_6 = \text{Dense}(4096, \text{activation}='relu')(\text{layer}_5) \]
8: \[ \text{layer}_7 = \text{Dense}(\text{input_dim}, \text{activation}='sigmoid')(\text{layer}_6) \]
9: \[ \text{model} = \text{Model}(\text{input} = \text{Layer}_1, \text{output} = \text{Layer}_7) \]  \( \triangleright \) create model by combining Layer 1 to Layer 7
10: \[ \text{model.compile(optimizer='adadelta', loss='binary_crossentropy')} \]
11: \[ \text{model.fit()} \]  \( \triangleright \) train the model
In practice, the dimension reduction did help improve the identification result a lot. Before dimension reduction, the Mean Square Error (MSE, more different, the better) between normal and abnormal samples are 0.005 vs 0.08. However, after dimension reduction, we get an average MSE=0.185 for same person’s profile (person 0) while MSE=3606.34 for person 1’s profile, which gives much better result.

3.4 The GAN Framework

The dataset we collected are huge. By careful examination, we notice that each user’s dataset also expose a strong trait of time-series property.

Many statistical prediction models have a goal of predicting a single quantitative outcome, for example, probability of 5-year cancer survival, or the number of wins of a sports team in a given season. Statistical models designed to predict the trajectory of a time series face added dimensions of complexity. For each observable unit of data (e.g., a time series observed up to a specific time), we might ask such models to predict not just one value, but a sequence of values. Additionally, if a robust and generalizable model is sought, the model must predict not just one time series effectively, but many.

These new dimensions quickly add complexity to the question of how to evaluate time series prediction models. If you are interested in evaluating predictions made at N separate time points, each at up to M time steps into the future, for L different time series, you need to make N \cdot M \cdot L distinct, if correlated, predictions.

Existing research has worked to identify the pros and cons of different methods for evaluating the accuracy of time series predictions. One trend in the literature highlights advantages of using relative absolute error metrics (e.g., the relative mean absolute error or the mean absolute scaled error) instead of squared error metrics to reduce the impact of outlying observations and to increase interpretability [83, 84]. In this context, several methods have been proposed to facilitate evaluation of predictions of seasonal data (see, e.g., [85],
although these methods do not appear to have been widely adopted. Additionally, the measure called “forecast skill,” which relies on a relative mean squared error calculation, has been widely used for several decades in the field of weather forecasting [86]. Another thread of work advocates for the use of proper scoring rules for probabilistic forecasts, where the observation is evaluated against the predicted distribution [87–89]. These methods, while having a strong theoretical foundation, are less directly comparable or interpretable and require a full predictive distribution.

Unfortunately, there are not too many works that utilize the semantic features learned by GANs to apply to anomaly detection tasks. Inspired by Yeh et al. [90], Schlegl et al. [91] represents one typical anomaly detection application that demonstrated the representative power of the generative model and the coupled mapping schema, which utilizes a trained DCGAN and enables accurate discrimination between normal anatomy, and local anomalous appearance. This renders the detection of subtle anomalies at scale feasible. However, as the dataset we have are large and sequential, such approaches are not suitable for sequential data.

The basic idea of GANs is to set up a game between two players. One of them is called the generator. The generator creates samples that are intended to come from the same distribution as the training data. The other player is the discriminator.

3.4.1 Discriminator

The discriminator examines samples to determine whether they are real or fake. The discriminator learns using traditional supervised learning techniques, dividing inputs into two classes (real or fake). The generator is trained to fool the discriminator. We can think of the generator as being like a counterfeiter, trying to make fake money, and the discriminator as being like police, trying to allow legitimate money and catch counterfeit money. To succeed in this game, the counterfeiter must learn to make money that is indistinguishable from genuine money, and the generator network must learn to create samples that are drawn from the same distribution as the training data. The process is illustrated in Fig. 3-8. The GAN framework pits two adversaries against each other in a game. Each player is represented by a differentiable
function controlled by a set of parameters. Typically these functions are implemented as deep neural networks. The game plays out in two scenarios. In one scenario, training examples $x$ are randomly sampled from the training set and used as input for the first player, the discriminator, represented by the function $D$. The goal of the discriminator is to output the probability that its input is real rather than fake, under the assumption that half of the inputs it is ever shown are real and half are fake. In this first scenario, the goal of the discriminator is for $D(x)$ to be near 1. In the second scenario, inputs $z$ to the generator are randomly sampled from the model’s prior over the latent variables. The discriminator then receives input $G(z)$, a fake
sample created by the generator. In this scenario, both players participate. The discriminator strives to make $D(G(z))$ approach 0 while the generative strives to make the same quantity approach 1. If both models have sufficient capacity, then the Nash equilibrium of this game corresponds to the $G(z)$ being drawn from the same distribution as the training data, and $D(x) = \frac{1}{2}$ for all $x$.

Formally, GANs are a structured probabilistic model [92] containing latent variables $z$ and observed variables $x$. The graph structure is shown in Fig. 3-9, which shows the graphical model structure of GANs, which is also shared with VAEs, sparse coding, etc. It is directed graphical model where every latent variable influences every observed variable. Some GAN variants remove some of these connections.

The two players in the game are represented by two functions, each of which is differentiable both with respect to its inputs and with respect to its parameters. The discriminator is a function $D$ that takes $x$ as input and uses $\theta^{(D)}$ as parameters. The generator is defined by a function $G$ that takes $z$ as input and uses $\theta^{(G)}$ as parameters.

Both players have cost functions that are defined in terms of both players’ parameters. The discriminator wishes to minimize $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ and must do so while controlling only

![Figure 3-9. The graphical model structure of GANs](image)
The generator wishes to maximize $J(D)(\theta^{(D)}, \theta^{(G)})$ and must do so while controlling only $\theta^{(G)}$. Because each player’s cost depends on the other player’s parameters, but each player cannot control the other player’s parameters, this scenario is most straightforward to describe as a game rather than as an optimization problem. The solution to an optimization problem is a (local) minimum, a point in parameter space where all neighboring points have greater or equal cost. The solution to a game is a Nash equilibrium. Here, we use the terminology of local differential Nash equilibria [93]. In this context, a Nash equilibrium is a tuple $(\theta^{(D)}, \theta^{(G)})$ that is a local minimum of $J(D)$ with respect to $\theta^{(D)}$ and a local minimum of $J(G)$ with respect to $\theta^{(G)}$.

### 3.4.1.1 Generator

The generator is simply a differentiable function $G$. When $z$ is sampled from some simple prior distribution, $G(z)$ yields a sample of $x$ drawn from $p_{model}$. Typically, a deep neural network is used to represent $G$. Note that the inputs to the function $G$ do not need to correspond to inputs to the first layer of the deep neural net; inputs may be provided at any point throughout the network. For example, we can partition $z$ into two vectors $z^{(1)}$ and $z^{(2)}$, then feed $z^{(1)}$ as input to the first layer of the neural net and add $z^{(2)}$ to the last layer of the neural net. If $z^{(2)}$ is Gaussian, this makes $x$ conditionally Gaussian given $z^{(1)}$. Another popular strategy is to apply additive or multiplicative noise to hidden layers or concatenate noise to hidden layers of the neural net. Overall, we see that there are very few restrictions on the design of the generator net. If we want $p_{model}$ to have full support on $x$ space we need the dimension of $z$ to be at least as large as the dimension of $x$, and $G$ must be differentiable, but those are the only requirements. In particular, note that any model that can be trained with the nonlinear ICA approach can be a GAN generator network. The relationship with variational autoencoders is more complicated; the GAN framework can train some models that the VAE framework cannot and vice versa, but the two frameworks also have a large intersection. The most salient difference is that, if relying on standard backprop, VAEs cannot have discrete
variables at the input to the generator, while GANs cannot have discrete variables at the output of the generator.

### 3.4.1.2 Training

The training process consists of simultaneous SGD. On each step, two minibatches are sampled: a minibatch of $x$ values from the dataset and a minibatch of $z$ values drawn from the model’s prior over latent variables. Then two gradient steps are made simultaneously: one updating $\theta^{(D)}$ to reduce $J^{(D)}$ and one updating $\theta^{(G)}$ to reduce $J^{(G)}$. In both cases, it is possible to use the gradient-based optimization algorithm of your choice. Adam [94] is usually a good choice. Many authors recommend running more steps of one player than the other, but as of late 2016, the author’s opinion is that the protocol that works the best in practice is simultaneous gradient descent, with one step for each player.

### 3.4.2 Cost Functions

There are several different cost functions used within the GANs framework.

#### 3.4.2.1 For discriminator

All of the different games designed for GANs so far use the same cost for the discriminator, $J^{(D)}$. They differ only in terms of the cost used for the generator, $J^{(G)}$. The cost used for the discriminator is:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2} \mathbb{E}_{x \sim \rho_{data}} \log D(x) - \frac{1}{2} \mathbb{E}_{x} \log(1 - D(G(z))). \quad (3-1)$$

This is just the standard cross-entropy cost that is minimized when training a standard binary classifier with a sigmoid output. The only difference is that the classifier is trained on two minibatches of data; one coming from the dataset, where the label is 1 for all examples, and one coming from the generator, where the label is 0 for all examples.

All versions of the GAN game encourage the discriminator to minimize Eq. (3-1). In all cases, the discriminator has the same optimal strategy.
3.4.2.2 For generator

So far we have specified the cost function for only the discriminator. A complete specification of the game requires that we specify a cost function also for the generator.

The simplest version of the game is a zero-sum game, in which the sum of all player’s costs is always zero. In this version of the game,

$$J^{(G)} = -J^{(D)}.$$  \hfill (3–2)

Because $J^{(G)}$ is tied directly to $J^{(D)}$, we can summarize the entire game with a value function specifying the discriminator’s payoff:

$$V(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)}).$$  \hfill (3–3)

Zero-sum games are also called minimax games because their solution involves minimization in an outer loop and maximization in an inner loop:

$$\theta^{(G)*} = \arg \min_{\theta^{(G)}} \max_{\theta^{(D)}} V(\theta^{(D)}, \theta^{(G)}).$$  \hfill (3–4)

The minimax game is mostly of interest because it is easily amenable to theoretical analysis. [15] used this variant of the GAN game to show that learning in this game resembles minimizing the Jensen-Shannon divergence between the data and the model distribution, and that the game converges to its equilibrium if both players’ policies can be updated directly in function space. In practice, the players are represented with deep neural nets and updates are made in parameter space, so these results, which depend on convexity, do not apply.

The cost used for the generator in the minimax game (Eq. (3–2)) is useful for theoretical analysis, but does not perform especially well in practice. Minimizing the cross-entropy between a target class and a classifier’s predicted distribution is highly effective because the cost never saturates when the classifier has the wrong output. The cost does eventually saturate, approaching zero, but only when the classifier has already chosen the correct class. In the minimax game, the discriminator minimizes a cross-entropy, but the generator maximizes the
same cross-entropy. This is unfortunate for the generator, because when the discriminator successfully rejects generator samples with high confidence, the generator’s gradient vanishes. To solve this problem, one approach is to continue to use cross-entropy minimization for the generator. Instead of flipping the sign on the discriminator’s cost to obtain a cost for the generator, we flip the target used to construct the cross-entropy cost. The cost for the generator then becomes:

$$J^{(G)} = -\frac{1}{2} \mathbb{E}_x \log D(G(z)).$$  \hspace{1cm} (3-5)

In the minimax game, the generator minimizes the log-probability of the discriminator being correct. In this game, the generator maximizes the log-probability of the discriminator being mistaken. This version of the game is heuristically motivated, rather than being motivated by a theoretical concern. The sole motivation for this version of the game is to ensure that each player has a strong gradient when that player is “losing” the game. In this version of the game, the game is no longer zero-sum, and cannot be described with a single value function. Other methods such as approximating maximum likelihood within the GANs framework are also possible cost function options [95].

3.4.2.3 Comparison of cost functions

We can think of the generator network as learning by a strange kind of reinforcement learning. Rather than being told a specific output $x$ it should associate with each $z$, the generator takes actions and receives rewards for them. In particular, note that $J^{(G)}$ does not make reference to the training data directly at all; all information about the training data comes only through what the discriminator has learned. (Incidentally, this makes GANs resistant to overfitting, because the generator has no opportunity in practice to directly copy training examples). The learning process differs somewhat from traditional reinforcement learning because

- The generator is able to observe not just the output of the reward function but also its gradients.
The reward function is non-stationary; the reward is based on the discriminator which learns in response to changes in the generator’s policy.

In all cases, we can think of the sampling process that begins with the selection of a specific \( z \) value as an episode that receives a single reward, independent of the actions taken for all other \( z \) values. The reward given to the generator is a function of a single scalar value, \( D(G(z)) \). We usually think of this in terms of cost (negative reward). The cost for the generator is always monotonically decreasing in \( D(G(z)) \) but different games are designed to make this cost decrease faster along different parts of the curve.

3.5 Bi-directional Attention GAN (AttenGAN)

Having showed the usefulness of GAN and how it works, in this section, we will describe how we employ the idea of GAN and propose a new framework to effectively solve our identification and anomaly detection tasks. We can see that existing models for identification and anomaly detection are still preliminary. We hypothesize that better performance can be obtained by incorporating knowledge from sequence time steps into the model. Therefore, we propose AttenGAN – a bi-directional sequential GAN that also uses attention weights. In the following sections, we first formally introduce generative adversarial nets (GAN) and then describe how our proposed AttenGAN works.

The overall architecture of the AttenGAN is shown in Fig. 3-10. It consists of two parts: a sequence encoder, a sequence attention layer. We describe the details of different components in the following sections.

3.5.1 Sequence Encoder

The AttenGAN sequence encoder is consisted of a bi-directional LSTM layer. Given a sequence with time step \( x_t, t \in [0, T] \), we use a bidirectional LSTM \([96]\) to get annotations of time steps by summarizing information from both directions for sequence, and therefore incorporate the contextual information in the annotation. The bidirectional LSTM contains the forward LSTM \( \overrightarrow{f} \) which reads the sequence \( x \) from \( x_0 \) to \( x_T \) and a backward LSTM \( \overleftarrow{f} \) which
reads from $x_t$ to $x_0$.

\[
\overrightarrow{h_t} = LSTM(x_t), \ t \in [0, T], \\
\overleftarrow{h_t} = LSTM(x_t), \ t \in [T, 0].
\] 

We obtain an annotation state $h_t$ for a given time step $x_t$ by concatenating the forward hidden state $\overrightarrow{h_t}$ and backward hidden state $\overleftarrow{h_t}$, i.e.,

\[
h_t = \begin{bmatrix} \overrightarrow{h_t} \\ \overleftarrow{h_t} \end{bmatrix}, \\
\] 

Figure 3-10. Illustration of AttenGAN model framework.
which summarizes the information of the whole sequence centered around time step $x_t$.

### 3.5.2 Sequence Attention Layer

Not all time steps in a sequence contribute equally to the representation of the sequence. Hence, we introduce attention mechanism to emphasize such time steps (receiving more weight) that are important to the characteristics of the sequence and aggregate the representation of those informative time steps to form a sequence. Specifically, we have the attention layer constructed as:

$$u_t = \tanh(W_x h_t + b_x), \quad (3-9)$$

$$\alpha_t = \frac{\exp(u_t^T \mu_s)}{\sum_{t=0}^{T} \exp(u_t^T \mu_s)}, \quad (3-10)$$

$$v = \sum_{t=0}^{T} \alpha_t h_t. \quad (3-11)$$

That is, we first feed the time step annotation $h_t$ through a one-layer MLP to get $u_t$ as a hidden representation of $h_t$, then we measure the importance of the sequence time step as the similarity of $u_t$ with a time step level context vector $\mu_s$ and get a normalized importance weight $\alpha_t$ through a softmax function. After that, we compute the sequence vector $v$ as a weighted sum of the time step annotations based on the weights. The context vector $\mu_s$ can be seen as a high level representation of a fixed query “what is the signature or informative time step” over the time steps like that used in memory networks [97, 98]. The time step context vector $\mu_s$ is randomly initialized and jointly learned during the training process.

### 3.5.3 Model Training

The sequence vector $v$ obtained from (3–11) can be regarded as a high level representation of the sequence as whole and can be used as features for sequence identification.

$$y = \text{softmax}(W_c v + b_c), \quad (3-12)$$

where $y$ is a $K$-dimensional vector $\{p_1, p_2, ..., p_K\}$. 
We can do semi-supervised learning with multi-class classifier by simply adding samples from the GAN generator $G$ to our data set, labeling them with a new “generated” class $y = K + 1$, and correspondingly increasing the dimension of our classifier output from $K$ to $K + 1$. We may then use $p_{\text{model}}(y = K + 1|x)$ to supply the probability that $x$ is fake, corresponding to $1 - D(x)$ in the original GAN framework. We can now also learn from unlabeled data, as long as we know that it corresponds to one of the $K$ classes of real data or one anomaly class by maximizing $\log p_{\text{model}}(y \in \{0, ..., K\}|x)$. Assuming half of our data set consists of real data and half of it is generated (this is arbitrary), our loss function for training the classifier then becomes

$$L = -E_{x,y \sim p_{\text{data}}(x,y)}(\log p_{\text{model}}(y|x)) - E_{x \sim G}(\log p_{\text{model}}(y = K + 1|x))$$

$$= L_1 + L_2,$$

where

$$L_1 = -E_{x,y \sim p_{\text{data}}(x,y)}(\log p_{\text{model}}(y|x, y < K + 1)),$$

$$L_2 = -E_{x \sim p_{\text{data}}(x)}(\log 1 - p_{\text{model}}(y = K + 1|x)) - E_{x \sim G}(\log p_{\text{model}}(y = K + 1|x)),$$

where we have decomposed the total cross-entropy loss into our standard supervised loss function $L_1$ (the negative log probability of the label, given that the data is real) and an unsupervised loss $L_2$ which is in fact the standard GAN game-value as becomes evident when we substitute $D(x) = 1 - p_{\text{model}}(y = K + 1|x)$ into the expression:

$$L_2 = -E_{x \sim p_{\text{data}}(x)}(\log D(x)) - E_{x \sim \text{noise}}(\log 1 - D(G(z))).$$

Then by minimizing both $L_1$ and $L_2$ iteratively, we are able to conduct the training process.

When testing and predicting, we are able to retrieve the class label $c$ by:

$$c = \arg \max_j p_j, j \in \{0, ..., K\}.$$
If \( c = 0 \), then the data are classified as anomaly. Further, if \( c \neq 0 \) and

\[
p_c < th,
\]

where \( th \) is certain threshold that lies between \([0, 1]\), then we also determine there is an anomaly; otherwise, the calculated class label \( c \) is the predicted class.

### 3.5.4 Preparing Training Data

One big challenge facing us all the time is the volume of our dataset: high-dimensional and memory consuming. In order to solve the high-dimensional problem, we have employed autoencoder based dimension reduction techniques in Section 3.3.4. Furthermore, in order to solve the memory consumption problem, we split the data series into segments with fixed sequence length (e.g., \( \text{length}=50 \)), and engage LSTM for training. This procedure is described in Algorithm 3.5.

**Algorithm 3.5. Preparing Training Data**

1. \textbf{procedure} \textsc{CreateTrainingData} (data\_matrix) \hspace{1em} \triangleright \textit{input is the feature vector matrix}
2. \hspace{1em} result = initialize to empty.
3. \hspace{1em} sequence\_length = 50
4. \hspace{1em} for each index \in index of data\_matrix do
5. \hspace{2em} result.append(data\_matrix[index : index + sequence\_length]) \hspace{1em} \triangleright \textit{split the dataset per sequence\_length}
6. \hspace{1em} end for
7. \hspace{1em} return result
8. \textbf{end procedure}

In practice, we also separate a small portion of the data for testing as well. That is, we use 90\% of the created data for training and the rest 10\% for testing.

### 3.6 Experiment Results

In this section, we will take a look at the experiment results of our proposed scheme. We will study metrics including identification accuracy, anomaly detection capability, convergence speed and attention weight distribution. First, we introduce our experiment setup and model configuration.
3.6.1 Model Configuration

By mapping features (File system, network and process) to binary vectors and concatenating them, we can build the sample data matrix for each user’s profile. As an example, for user 0, we have 4290 samples and each sample is 111569 dimension. We build the training network structure in a deep neural network GAN fashion. The discriminator consists of our proposed AttenGAN layer and one dense layer for output. The structure framework is illustrated as in Fig. 3-10.

The dataset consists of data collected from 3 real persons. For each person, we collected their activity at 1 minute interval, that is, we have a time series data resolution of 1 minute per record. The total length of time for these data collection lasts 3 weeks. The system works by taking a time series data of 50 steps that are of 1024 dimensions each. The reason why the input is 1024 instead of original 111k+ dimensional data is attributed to the dimension reduction mechanism we proposed in Section 3.3. The input is dimension reduced data from Algorithm 3.4. The generator takes input noise of dimension 100.

The training framework involves a generator and a discriminator and they both train iteratively. We build the system using Tensorflow [99]. The generator is consisted of a deep LSTM layer and a dense layer for output. The LSTM layer consists of 300 LSTM units. The bi-directional LSTM in discriminator is consisted of two LSTM layers with each of 300 LSTM units.

The output is the class probability of which the input class belongs. This is achieved by taking the output of AttenGAN layer and let it go through a softmax layer. We get the index of the maximum probability as the predicted class label for the input data or anomaly detected, as described in Section 3.5.3.

3.6.2 User Profile Identification Accuracy

User profile identification accuracy measures how accurate the model can make identification by testing it on a given dataset. It has relation with number of correctly predicted class with total number of samples. The identification accuracy $\rho$ given a model
is calculated by:
\[ p = \frac{\text{Number of correctly predicted class}}{\text{total number of samples}}. \] (3–20)

In this experiment, we consider 3 real persons’ profiles labeled as class 1, 2, and 3, combined with one anomaly class labeled as 0. By number of correctly predicted class, we consider either the 3 user profiles are correctly predicted (as class 1, 2 or 3), or the anomaly class is correctly detected (as class 0).

Fig. 3-11 shows the relation between identification accuracy and number of epochs. There are 4 schemes that are in comparison which include our proposed AttenGAN, a type of sequential GAN (SeqGAN) [100], a simple LSTM classifier and a classical deep neural network.

Figure 3-11. AttenGAN identification accuracy compared with other schemes.
Table 3-1. AttenGAN accuracy comparison at Epoch 30.

<table>
<thead>
<tr>
<th></th>
<th>AttenGAN</th>
<th>SeqGAN</th>
<th>LSTM</th>
<th>DNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.988</td>
<td>0.941</td>
<td>0.911</td>
<td>0.867</td>
</tr>
</tbody>
</table>

(DNN) classifier. Note that the discriminator of SeqGAN is a modified version that can output classes instead of simple 0/1 output.

From these plots, we are able to make the following observations. First, the accuracy measurement of all of these approaches increases during training. That is, as the number of epochs increases, the accuracy increases accordingly. Second, thanks to the attention mechanism, our proposed AttenGAN achieves the best accuracy performance among all, followed by SeqGAN and LSTM. DNN achieves the worst accuracy performance. This fact can be further justified by taking a look at the specific accuracy value at epoch 30, shown in Table 3-1. The reason is obvious, as both SeqGAN and LSTM are not able to capture the attention information in our sequence. To make things worse, DNN even cannot capture the time series information. We put it here simply for comparison purposes. Third, we can see that two GAN based scheme – AttenGAN and SeqGAN – generally perform better than classical neural network schemes – LSTM and DNN. This further justifies our hypothesis that GAN has the potential to build strong and better classifiers as well.

### 3.6.3 Anomaly Detection Capability

Anomaly detection capability is often measured in receiver operating characteristic (ROC) curve. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. True positive rate measures the proportion of positives that are correctly identified as such and false positive rate is the probability of falsely rejecting the null hypothesis for a particular test. The false positive rate is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events (regardless of identification, i.e., anomalies).

In this experiment, we only consider two classes. One is anomaly class, the other is normal class which corresponds to the specific user profile identification that does not identified as
anomaly. By setting different value of identification probability threshold, we are able to plot the ROC curve for our 3 comparison schemes: AttenGAN, SeqGAN and LSTM. Again, the SeqGAN is specifically modified to adapt to identification tasks. The results are shown in Fig. 3-12.

![AttenGAN ROC curve compared with other schemes.](image)

From these plots, we are able to make the following observations. First, the TPR measurement of all of these approaches increases accordingly as the FPR increases. Second, thanks to the attention mechanism, our proposed AttenGAN is able to achieve the best TPR performance among all at given FPR value, followed by SeqGAN and LSTM. The reason is obvious, as both SeqGAN and LSTM are not able to capture the attention information in our sequence. Third, we can see that two GAN based scheme – AttenGAN and SeqGAN –
generally perform better than classical LSTM scheme. This further justifies our hypothesis that GAN has the potential to build strong and better anomaly detector as well.

3.6.4 Convergence Speed

Another interesting fact we are interested in is how fast the learning algorithm learns. This can be measured by the non-negative log likelihood (NLL) after a number of epochs. We call this convergence speed. The NLL is calculated by:

\[ NLL = -\mathbb{E} \left\{ \sum_d \log p_{dj} \right\}, \quad (3-21) \]

where \( j \) is the label of sequence \( d \) and \( p \) is the probability that \( d \) is predicted to belong to label \( j \).

As illustrated in Fig. 3-13, we plot the curves that represent the relations between

![Figure 3-13. AttenGAN convergence speed comparison with other schemes.](image-url)
number of epochs for training and the NLL loss value.

From these plots, we are able to make the following observations. First, all the schemes are converging fast in the first few epochs (less than 5) of training. This is because all the internal states were randomly set when the system initializes. After that, the system’s learning speed would slow down and finally reach a stable state at around epoch 15. This is an expected behavior. Second, thanks to the attention mechanism, our proposed AttenGAN is able to achieve the fastest convergence speed among all (blue line), followed by SeqGAN and LSTM. Furthermore, it looks like the attention mechanism also makes AttenGAN achieve better convergence performance in terms of NLL loss value around epoch 30. The reason is obvious, as both SeqGAN and LSTM are not able to capture the attention information in our sequence. This further justifies our hypothesis that GAN has the potential to build strong and better anomaly detector as well. Third, DNN performs worst. This is a good old approach which does not have too much performance advantage when compared with more advanced GAN based approaches.

3.6.5 Attention Weight Distribution

Regarding the attention mechanism of AttenGAN, we are also interested in seeing how the attention weight distribution looks like in practice. The attention weights we are tracking are those $\alpha_i$ in (3–10). If certain time steps were inherently important or not important, models without attention mechanism might work well since the model could automatically assign low weights to irrelevant time steps and vice versa. However, the importance of certain time steps is highly context dependent.

To verify that our model can capture context dependent time step importance, we plot the distribution of the attention weights of sequence at time step 0, 15, 30 and 45 respectively for 3 user profiles numbered at 1, 2, 3 which are shown in Fig. 3-14. From these plots, we can see that for profile #1, the attention weights tend to decrease as the time step increases; for profile #3, the attention weights tend to increase as time step increases. Second, different
user profiles show different distribution pattern. The weights are not distributed evenly. This fact does show the effectiveness of our attention mechanism as different time steps would have different importance weight along the sequence.

3.7 Chapter Summary

In this chapter, we have proposed a bi-directional attention GAN (AttenGAN) based system for user profile anomaly detection. Due to the large volume of our collected dataset, the designed system is able to first reduce the dimension of the original dataset by a carefully
designed autoencoder based dimension reduction mechanism, and then a GAN based identification called AttenGAN was proposed for later identification tasks. The proposed framework works very well with respect to the volume of our dataset. The detection accuracy is high and the results satisfy the required standard set by our project goals.
CHAPTER 4
PARTIAL LEARNING GENERATIVE ADVERSARIAL NETWORKS FOR USER PROFILE IDENTIFICATION AND ANOMALY DETECTION

In this chapter, we study a more challenging problem than that raised in Chapter 3. We will describe a more advanced approach for solving the user profile identification and anomaly detection problem. The proposed approach employs a technique called “Partial GAN” or P-GAN, which can utilize partially generated sequence and internal state to help GAN make decision.

4.1 Problem Formulation

In Chapter 3, we described how we solved the user profile anomaly detection problem using a series of autoencoder based dimension reduction techniques and proposed a novel AttenGAN based identification and anomaly detection framework. With all the satisfactory results, however, one key problem remains unsolved. That is, the AttenGAN framework as well as all the existing sequence GAN frameworks requires a fixed length of time step $T$ to enable discriminator to make decision. This requirement enforces a strong limitation on the applicability of existing framework. As the lack of interaction between discriminator and generator prevails, the feedback delay from discriminator to generator during the learning process keeps long. The fixed length of time step also affects the adaptability of the system.

In this chapter, we study the more challenging problem. We propose a user profile identification and anomaly detection scheme that is able to capture the partial sequence information and hence can make decisions in the middle of sequence. Different from all the existing works [100], the proposed scheme, for the first time, uses partial internal state to facilitate sequential decision making process. It does not require a fixed length of time series for training and the length of sequence can be dynamic. Hence, it can significantly reduce feedback delay from discriminator to generator from $T$ time slots to 1 time slot and greatly improves the adaptivity of system to dynamic time series where a fixed time step $T$ is difficult to determine.
In the following sections, we follow [101, 102] and consider the sequence generation procedure as a sequential decision making process. The proposed approach is based on GAN and LSTM. Let’s first start out with an introduction of related models.

4.2 Related Models

GANs [103] enable to learn generative models generating detailed realistic images [15, 104, 105]. Radford et al. [106] introduced deep convolutional generative adversarial networks (DCGANs) and showed that GANs are capable of capturing semantic image content enabling vector arithmetic for visual concepts. Yeh et al. [90] trained GANs on natural images and applied the trained model for semantic image inpainting. Unfortunately, there are not too many works that utilize the semantic features learned by GANs to apply to anomaly detection tasks. Inspired by Yeh et al. [90], Schlegl et al. [91] represents one typical anomaly detection application that demonstrated the representative power of the generative model and the coupled mapping schema, which utilizes a trained DCGAN and enables accurate discrimination between normal anatomy, and local anomalous appearance. This renders the detection of subtle anomalies at scale feasible. However, as the dataset we have are large and sequential, such approaches are not suitable for sequential data.

Following the introduction in Chapter 1, we can have some idea of what generative models can do and why it might be desirable to build one. Now we can ask: how does a generative model actually work? And in particular, how does a GAN work, in comparison to other generative models?

4.2.1 Maximum Likelihood Estimation

To simplify the discussion somewhat, we will focus on generative models that work via the principle of maximum likelihood. Not every generative model uses maximum likelihood. Some generative models do not use maximum likelihood by default, but can be made to do so (GANs fall into this category). By ignoring those models that do not use maximum likelihood, and by focusing on the maximum likelihood version of models that do not usually use maximum likelihood, we can eliminate some of the more distracting differences between different models.
The basic idea of maximum likelihood is to define a model that provides an estimate of a probability distribution, parameterized by parameters \( \theta \). We then refer to the likelihood as the probability that the model assigns to the training data: \( \prod_{i=1}^{m} p_{\text{model}}(x^{(i)}; \theta) \), for a dataset containing \( m \) training examples \( x^{(i)} \).

The principle of maximum likelihood simply says to choose the parameters for the model that maximize the likelihood of the training data. This is easiest to do in log space, where we have a sum rather than a product over examples. This sum simplifies the algebraic expressions for the derivatives of the likelihood with respect to the models, and when implemented on a digital computer, is less prone to numerical problems, such as underflow resulting from multiplying together several very small probabilities.

\[
\theta^* = \arg \max_{\theta} \prod_{i=1}^{m} p_{\text{model}}(x^{(i)}; \theta) \tag{4-1}
\]

\[
= \arg \max_{\theta} \log \prod_{i=1}^{m} p_{\text{model}}(x^{(i)}; \theta) \tag{4-2}
\]

\[
= \arg \max_{\theta} \sum_{i=1}^{m} \log p_{\text{model}}(x^{(i)}; \theta). \tag{4-3}
\]

In Eq. (4–2), we have used the property that \( \arg \max_{v} f(v) = \arg \max_{v} \log f(v) \) for positive \( v \), because the logarithm is a function that increases everywhere and does not change the location of the maximum. As illustrated in Fig. 4-1, the maximum likelihood process consists of taking several samples from the data generating distribution to form a training set, then pushing up on the probability the model assigns to those points, in order to maximize the likelihood of the training data. This illustration shows how different data points push up on different parts of the density function for a Gaussian model applied to 1-D data. The fact that the density function must sum to 1 means that we cannot simply assign infinite likelihood to all points; as one point pushes up in one place it inevitably pulls down in other places. The resulting density function balances out the upward forces from all the data points in different locations.
We can also think of maximum likelihood estimation as minimizing the KL divergence between the data generating distribution and the model:

\[ \theta^* = \arg\min_{\theta} D_{KL}(p_{data}(x) \| p_{model}(x; \theta)) \]  

(4-4)

If we were able to do this precisely, then if \( p_{data} \) lies within the family of distributions \( p_{model}(x; \theta) \), the model would recover \( p_{data} \) exactly. In practice, we do not have access to \( p_{data} \) itself, but only to a training set consisting of \( m \) samples from \( p_{data} \). We use these to define \( \hat{p}_{data} \), an empirical distribution that places mass only on exactly those \( m \) points, approximating \( p_{data} \). Minimizing the KL divergence between \( \hat{p}_{data} \) and \( p_{model} \) is exactly equivalent to maximizing the log-likelihood of the training set.

For more information on maximum likelihood and other statistical estimators, see chapter 5 of [92].

### 4.2.2 A Taxonomy of Deep Generative Models

If we restrict our attention to deep generative models that work by maximizing the likelihood, we can compare several models by contrasting the ways that they compute either the likelihood and its gradients, or approximations to these quantities. As mentioned earlier,
many of these models are often used with principles other than maximum likelihood, but we can examine the maximum likelihood variant of each of them in order to reduce the amount of distracting differences between the methods. Following this approach, we construct the taxonomy shown in Fig. 4-2. On the left branch of this taxonomic tree, models construct an explicit density, \( p_{\text{model}}(x; \theta) \), and thus an explicit likelihood which can be maximized. Among these explicit density models, the density may be computationally tractable, or it may be intractable, meaning that to maximize the likelihood it is necessary to make either variational approximations or Monte Carlo approximations (or both). On the right branch of the tree, the model does not explicitly represent a probability distribution over the space where the data lies. Instead, the model provides some way of interacting less directly with this probability distribution. Typically the indirect means of interacting with the probability distribution is the ability to draw samples from it. Some of these implicit models that offer the ability to sample from the distribution do so using a Markov Chain; the model defines a way to stochastically transform an existing sample in order to obtain another sample from the same distribution. Others are able to generate a sample in a single step, starting without any input. While the

![Figure 4-2. Deep generative models that can learn via the principle of maximum likelihood differ with respect to how they represent or approximate the likelihood.](image-url)
models used for GANs can sometimes be constructed to define an explicit density, the training algorithm for GANs makes use only of the model’s ability to generate samples. GANs are thus trained using the strategy from the rightmost leaf of the tree: using an implicit model that samples directly from the distribution represented by the model.

Every leaf in this taxonomic tree has some advantages and disadvantages. GANs were designed to avoid many of the disadvantages present in pre-existing nodes of the tree, but also introduced some new disadvantages.

4.2.3 Explicit Density Models

In the left branch of the taxonomy shown in Fig. 4-2 are models that define an explicit density function $p_{model}(x; \theta)$. For these models, maximization of the likelihood is straightforward; we simply plug the model’s definition of the density function into the expression for the likelihood, and follow the gradient uphill. The main difficulty present in explicit density models is designing a model that can capture all of the complexity of the data to be generated while still maintaining computational tractability. There are two different strategies used to confront this challenge: (1) careful construction of models whose structure guarantees their tractability, as described in Section 4.2.3, and (2) models that admit tractable approximations to the likelihood and its gradients, as described in Section 4.2.3.

In the leftmost leaf of the taxonomic tree of Fig. 4-2 are the models that define an explicit density function that is computationally tractable. There are currently two popular approaches to tractable explicit density models: fully visible belief networks and nonlinear independent components analysis.

Chapter 3 of [92] provides more detailed information about the chain rule of probability used to define FVBNs or about the effect of deterministic transformations on probability densities as used to define nonlinear ICA models.

In summary, models that define an explicit, tractable density are highly effective, because they permit the use of an optimization algorithm directly on the log-likelihood of the training
data. However, the family of models that provide a tractable density is limited, with different families having different disadvantages.

To avoid some of the disadvantages imposed by the design requirements of models with tractable density functions, other models have been developed that still provide an explicit density function but use one that is intractable, requiring the use of approximations to maximize the likelihood. These fall roughly into two categories: those using deterministic approximations, which almost always means variational methods, and those using stochastic approximations, meaning Markov chain Monte Carlo methods.

**Variational approximations** Variational methods define a lower bound

\[
L(x; \theta) \leq \log p_{\text{model}}(x; \theta).
\]

A learning algorithm that maximizes \(L\) is guaranteed to obtain at least as high a value of the log-likelihood as it does of \(L\). For many families of models, it is possible to define an \(L\) that is computationally tractable even when the log-likelihood is not. Currently, the most popular approach to variational learning in deep generative models is the variational autoencoder \([107]\) or VAE. Variational autoencoders are one of the three approaches to deep generative modeling that are the most popular as of this writing, along with FVBNs and GANs. The main drawback of variational methods is that, when too weak of an approximate posterior distribution or too weak of a prior distribution is used, even with a perfect optimization algorithm and infinite training data, the gap between \(L\) and the true likelihood can result in pmodel learning something other than the true pdata. GANs were designed to be unbiased, in the sense that with a large enough model and infinite data, the Nash equilibrium for a GAN game corresponds to recovering pdata exactly. In practice, variational methods often obtain very good likelihood, but are regarded as producing lower quality samples. There is not a good method of quantitatively measuring sample quality, so this is a subjective opinion, not an empirical fact. While it is difficult to point to a single aspect of GAN design and say that it results in better sample quality,
GANs are generally regarded as producing better samples. Compared to FVBNs, VAEs are regarded as more difficult to optimize, but GANs are not an improvement in this respect.

**Markov chain approximations** Most deep learning algorithms make use of some form of stochastic approximation, at the very least in the form of using a small number of randomly selected training examples to form a minibatch used to minimize the expected loss. Usually, sampling-based approximations work reasonably well as long as a fair sample can be generated quickly (e.g. selecting a single example from the training set is a cheap operation) and as long as the variance across samples is not too high. Some models require the generation of more expensive samples, using Markov chain techniques. A Markov chain is a process for generating samples by repeatedly drawing a sample $x' \sim q(x'|x)$. By repeatedly updating $x$ according to the transition operator $q$, Markov chain methods can sometimes guarantee that $x$ will eventually converge to a sample from $p_{model}(x)$. Unfortunately, this convergence can be very slow, and there is no clear way to test whether the chain has converged, so in practice one often uses $x$ too early, before it has truly converged to be a fair sample from $p_{model}$. In high-dimensional spaces, Markov chains become less efficient. Boltzmann machines [108–111] are a family of generative models that rely on Markov chains both to train the model or to generate a sample from the model. Boltzmann machines were an important part of the deep learning renaissance beginning in 2006 [75, 112] but they are now used only very rarely, presumably mostly because the underlying Markov chain approximation techniques have not scaled to problems like ImageNet generation. Moreover, even if Markov chain methods scaled well enough to be used for training, the use of a Markov chain to generate samples from a trained model is undesirable compared to single-step generation methods because the multi-step Markov chain approach has higher computational cost. GANs were designed to avoid using Markov chains for these reasons.
Some models use both variational and Markov chain approximations. For example, deep Boltzmann machines make use of both types of approximation [113].

4.2.4 Implicit Density Models

Some models can be trained without even needing to explicitly define a density functions. These models instead offer a way to train the model while interacting only indirectly with \( p_{\text{model}} \), usually by sampling from it. These constitute the second branch, on the right side, of our taxonomy of generative models depicted in Fig. 4-2.

Some of these implicit models based on drawing samples from \( p_{\text{model}} \) define a Markov chain transition operator that must be run several times to obtain a sample from the model. From this family, the primary example is the generative stochastic network [114]. As discussed in Section 4.2.3, Markov chains often fail to scale to high dimensional spaces, and impose increased computational costs for using the generative model. GANs were designed to avoid these problems.

Finally, the rightmost leaf of our taxonomic tree is the family of implicit models that can generate a sample in a single step. At the time of their introduction, GANs were the only notable member of this family, but since then they have been joined by additional models based on kernelized moment matching [115, 116].

4.2.5 Comparing Generative Models

In summary, GANs were designed to avoid many disadvantages associated with other generative models:

- They can generate samples in parallel, instead of using runtime proportional to the dimensionality of \( \mathbf{x} \). This is an advantage relative to FVBNs.

- The design of the generator function has very few restrictions. This is an advantage relative to Boltzmann machines, for which few probability distributions admit tractable Markov chain sampling, and relative to nonlinear ICA, for which the generator must be invertible and the latent code \( \mathbf{z} \) must have the same dimension as the samples \( \mathbf{x} \).

- No Markov chains are needed. This is an advantage relative to Boltzmann machines and GSNs.
- No variational bound is needed, and specific model families usable within the GAN framework are already known to be universal approximators, so GANs are already known to be asymptotically consistent. Some VAEs are conjectured to be asymptotically consistent, but this is not yet proven.

- GANs are subjectively regarded as producing better samples than other methods. At the same time, GANs have taken on a new disadvantage: training them requires finding the Nash equilibrium of a game, which is a more difficult problem than optimizing an objective function.

At the same time, GANs have taken on a new disadvantage: training them requires finding the Nash equilibrium of a game, which is a more difficult problem than optimizing an objective function.

4.2.6 Sequential GAN

In previous chapters and sections, we have described that GANs are useful and capable of generating synthetic data through a buildup of deep neural networks or convolutional networks \[106\]. However, generating sequential synthetic data that mimics the real one is an important problem in unsupervised learning. Recently, recurrent neural networks (RNNs) with long short-term memory (LSTM) cells \[117\] have shown excellent performance ranging from natural language generation to handwriting generation \[118, 119\]. The most common approach to training an RNN is to maximize the log predictive likelihood of each true token in the training sequence given the previous observed tokens \[113\]. However, as argued in \[120\], the maximum likelihood approaches suffer from so-called exposure bias in the inference stage: the model generates a sequence iteratively and predicts next token conditioned on its previously predicted ones that may be never observed in the training data. Such a discrepancy between training and inference can incur accumulatively along with the sequence and will become prominent as the length of sequence increases. To address this problem, \[120\] proposed a training strategy called scheduled sampling (SS), where the generative model is partially fed with its own synthetic data as prefix (observed tokens) rather than the true data when deciding the next token in the training stage. Nevertheless, \[121\] showed that SS is an inconsistent training strategy and fails to address the problem fundamentally. Another possible solution of
the training/inference discrepancy problem is to build the loss function on the entire generated sequence instead of each transition. For instance, in the application of machine translation, a task specific sequence score/loss, bilingual evaluation understudy [122] can be adopted to guide the sequence generation. However, in many other practical applications, such as poem generation [123] and chatbot [124], a task specific loss may not be directly available to score a generated sequence accurately.

GANs provide good way to alleviate the problem above. Specifically, in GAN a discriminative net D learns to distinguish whether a given data instance is real or not, and a generative net G learns to confuse D by generating high quality data. This approach has been successful and been mostly applied in computer vision tasks of generating samples of natural images [104].

Unfortunately, applying GAN to generating sequences has two problems. Firstly, GAN is designed for generating real-valued, continuous data but has difficulties in directly generating sequences of discrete tokens, such as texts [121]. The reason is that in GANs, the generator starts with random sampling first and then a deterministic transform, governed by the model parameters. As such, the gradient of the loss from D w.r.t. the outputs by G is used to guide the generative model G (parameters) to slightly change the generated value to make it more realistic. If the generated data is based on discrete tokens, the “slight change” guidance from the discriminative net makes little sense because there is probably no corresponding token for such slight change in the limited dictionary space [103]. Secondly, GAN can only give the score/loss for an entire sequence when it has been generated; for a partially generated sequence, it is non-trivial to balance how good as it is now and the future score as the entire sequence.

The GANs framework bypasses the difficulty of maximum likelihood learning and has gained striking successes in natural image generation [104]. However, little progress has been made in applying GANs to sequence discrete data generation problems, e.g. natural language generation [121]. This is due to the generator network in GAN is designed to be able to adjust the output continuously, which does not work on discrete data generation [103].
On the other hand, a lot of efforts have been made to generate structured sequences. Recurrent neural networks can be trained to produce sequences of tokens in many applications such as machine translation [125, 126]. The most popular way of training RNNs is to maximize the likelihood of each token in the training data whereas [120] pointed out that the discrepancy between training and generating makes the maximum likelihood estimation suboptimal and proposed scheduled sampling strategy (SS). Later [121] theorized that the objective function underneath SS is improper and explained the reason why GANs tend to generate natural-looking samples in theory. Consequently, the GANs have great potential but are not practically feasible to discrete probabilistic models currently.

As pointed out by [101], the sequence data generation can be formulated as a sequential decision making process, which can be potentially be solved by reinforcement learning techniques. Modeling the sequence generator as a policy of picking the next token, policy gradient methods [127] can be adopted to optimize the generator once there is an (implicit) reward function to guide the policy. For most practical sequence generation tasks, e.g. machine translation [125], the reward signal is meaningful only for the entire sequence, for instance in the game of Go [128], the reward signal is only set at the end of the game. In those cases, state-action evaluation methods such as tree search have been adopted [129].

4.3 Review of LSTM

As “Long Short-Term Memory” (LSTM) plays an important part in our proposed model, a brief review is presented in the following paragraphs.

Recurrent Neural Network (RNN) constitute a very powerful class of computational models, capable of instantiating almost arbitrary dynamics. The extent to which this potential can be exploited, is however limited by the effectiveness of the training procedure applied. Gradient based methods – “Back-Propagation Through Time” [130, 131] or – “Real-Time Recurrent Learning” [130, 132] share an important limitation. The temporal evolution of the path integral over all error signals “flowing back in time” exponentially depends on the magnitude of the weights [133]. This implies that the backpropagated error quickly either
vanishes or blows up [117, 134]. Hence standard RNNs fail to learn in the presence of time lags greater than 5 - 10 discrete time steps between relevant input events and target signals. The vanishing error problem casts doubt on whether standard RNNs can indeed exhibit significant practical advantages over time window-based feedforward networks.

A newer RNN model, LSTM [117] has emerged as an effective and scalable model for several learning problems related to sequential data. Earlier methods for attacking these problems have either been tailored toward a specific problem or did not scale to long time dependences. LSTMs on the other hand are both general and effective at capturing long-term temporal dependences. They do not suffer from the optimization hurdles that plague simple recurrent networks (SRNs) [135] and have been used to advance the state of the art for many difficult problems. This includes handwriting recognition [136–138] and generation [119], language modeling [139] and translation [140], acoustic modeling of speech [141], speech synthesis [142], protein secondary structure prediction [143], analysis of audio [144], and video data [105] among others.

LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing constant error flow through “constant error carrousels” (CECs) within special units, called cells. Multiplicative gate units learn to open and close access to the cells. LSTM’s learning algorithm is local in space and time; its computational complexity per time step and weight is O(1). It solves complex long time lag tasks that have never been solved by previous RNN algorithms.

The core idea behind the LSTM architecture is a memory cell, which can maintain its state over time, and nonlinear gating units, which regulate the information flow into and out of the cell. Most modern studies incorporate many improvements that have been made to the LSTM architecture since its original formulation [117]. However, LSTMs are now applied to many learning problems, which differ significantly in scale and nature from the problems that these improvements were initially tested on. The architecture of computational components of LSTM is shown in Fig. 4-3.
The initial version of the LSTM block [117] included (possibly multiple) cells and input and output gates, but no forget gate (NFG) and no peephole (NP) connections. The output gate, unit biases, or input activation function were omitted for certain experiments. Training was done using a mixture of real-time recurrent learning [132, 145] and backpropagation through time (BPTT) [131, 145]. Only the gradient of the cell was propagated back through time, and the gradient for the other recurrent connections was truncated. Thus, that study did not use the exact gradient for training. Another feature of that version was the use of full gate recurrence (FGR), which means that all the gates received recurrent inputs from all the gates at the previous time step in addition to the recurrent inputs from the block outputs. This feature did not appear in any of the later papers.

**Forget Gate** The first paper to suggest a modification of the LSTM architecture introduced the forget gate [146], enabling the LSTM to reset its own state. This allowed learning of continual tasks such as embedded Reber grammar.

**Peephole Connections** Gers and Schmidhuber [147] argued that in order to learn precise timings, the cell needs to control the gates. So far, this was only possible through an
open output gate. Peephole connections (connections from the cell to the gates as in Fig. 4-3 (blue curves)) were added to the architecture in order to make precise timings easier to learn. In addition, the output activation function was omitted, as there was no evidence that it was essential for solving the problems that LSTM had been tested on so far.

**Full Gradient** The final modification toward the vanilla LSTM was done by Graves and Schmidhuber [148]. This paper presented the full BPTT training for LSTM networks with the architecture described in Section II, and presented results on the TIMIT [149] benchmark. Using full BPTT had the added advantage that LSTM gradients could be checked using finite differences, making practical implementations more reliable.

### 4.3.1 Vanilla LSTM

The LSTM setup most commonly used in the literature was originally described by Graves and Schmidhuber [148]. We refer to it as vanilla LSTM and use it as a reference for comparison of all the variants. The vanilla LSTM incorporates changes by Gers et al. [146] and Gers and Schmidhuber [147] into the original LSTM [117] and uses full gradient training. Section III provides descriptions of these major LSTM changes.

A schematic of the vanilla LSTM block can be seen in Fig. 4-3. It features three gates (input, forget, and output), block input, a single cell (the constant error carousel), an output activation function, and peephole connections

\[ 1 \] . The output of the block is recurrently connected back to the block input and all of the gates [150].

### 4.3.2 Forward pass

Let \( \mathbf{x}^t \) be the input vector at time \( t \), \( N \) be the number of LSTM blocks, and \( M \) the number of inputs. Then, we get the following weights for an LSTM layer.

- **Input weights**: \( \mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{N \times M} \).

\[ 1 \] Some studies omit peephole connections, see Section 4.3.
• Recurrent weights: $R_z, R_s, R_f, R_o \in \mathbb{R}^{N \times N}$.

• Peehole weights: $p_s, p_f, p_o \in \mathbb{R}^N$.

• Bias weights: $b_z, b_s, b_f, b_o \in \mathbb{R}^N$.

Then the vector formulas for a vanilla LSTM layer forward pass can be written as

\[
\begin{align*}
\tilde{z}^i &= W_z x^i + R_z y^{i-1} + b_z & (4-6) \\
z^i &= g(\tilde{z}^i) & \text{block input} & (4-7) \\
\tilde{i}^i &= W_i x^i + R_i y^{i-1} + p_i \odot c^{i-1} b_i & (4-8) \\
i^i &= \sigma(\tilde{i}^i) & \text{input gate} & (4-9) \\
\tilde{f}^i &= W_f x^i + R_f y^{i-1} + p_f \odot c^{i-1} b_f & (4-10) \\
f^i &= \sigma(\tilde{f}^i) & \text{forget gate} & (4-11) \\
c^i &= z^i \odot i^i + c^{i-1} \odot f^i & \text{cell} & (4-12) \\
\tilde{o}^i &= W_o x^i + R_o y^{i-1} + p_o \odot c^{i-1} b_o & (4-13) \\
o^i &= \sigma(\tilde{o}^i) & \text{output gate} & (4-14) \\
y^i &= h(c^i) \odot o^i & \text{block output} & (4-15)
\end{align*}
\]

where $\sigma$, $g$, and $h$ are pointwise nonlinear activation functions. The logistic sigmoid ($\sigma(x) = (1/1 + e^{-x})$) is used as the gate activation function and the hyperbolic tangent ($g(x) = h(x) = tanh(x)$) is usually used as the block input and output activation function. Pointwise multiplication of two vectors is denoted by $\odot$. 
4.3.3 Backpropagation through time

The deltas inside the LSTM block are then calculated as

$$\delta y^t = \Delta^t + R_z^T \delta z^{t+1} + R_i^T \delta i^{t+1} + R_f^T \delta f^{t+1} + R_o^T \delta o^{t+1}$$  \hspace{1cm} (4–16)

$$\delta o^t = \delta y^t \odot h(c^t) \odot \sigma'(\delta^t)$$ \hspace{1cm} (4–17)

$$\delta c^t = \delta y^t \odot o^t \odot h'(c^t) + p_o \odot \delta o^t + p_i \odot \delta i^{t+1} + p_f \odot \delta f^{t+1} + \delta c^{t+1} \odot f^{t+1}$$  \hspace{1cm} (4–18)

$$\delta f^t = \delta c^t \odot c^{t-1} \odot \delta'(\tilde{f}^t)$$ \hspace{1cm} (4–19)

$$\delta i^t = \delta c^t \odot z^{t-1} \odot \delta'(\tilde{i}^t)$$ \hspace{1cm} (4–20)

$$\delta z^t = \delta c^t \odot i^{t-1} \odot g'(\tilde{z}^t).$$ \hspace{1cm} (4–21)

Here, $\Delta^t$ is the vector of the deltas passed down from the layer above. If $E$ is the loss function, it formally corresponds to $\partial E / \partial y^t$, but not including the recurrent dependences. The deltas for the inputs are only needed if there is a layer below that needs training, and can be computed as follows:

$$\delta x^t = W_z^T \delta z^t + W_i^T \delta i^t + W_f^T \delta f^t + W_o^T \delta o^t.$$ \hspace{1cm} (4–22)

Finally, the gradients for the weights are calculated as follows, where $\ast$ can be any of $\{z, i, f, o\}$, and $\langle a, b \rangle$ denotes the outer product of two vectors $a$ and $b$:

$$\delta W_\ast = \sum_{t=0}^{T} \langle \delta x^t, x^t \rangle \delta p_\ast = \sum_{t=0}^{T-1} c^t \odot \delta i^{t+1}$$ \hspace{1cm} (4–23)

$$\delta R_\ast = \sum_{t=0}^{T-1} \langle \delta s^{t+1}, y^t \rangle \delta p_f = \sum_{t=0}^{T-1} c^t \odot \delta i^{t+1}$$ \hspace{1cm} (4–24)

$$\delta b_\ast = \sum_{t=0}^{T} \delta x^t \delta p_\ast = \sum_{t=0}^{T} c^t \odot \delta o^t.$$ \hspace{1cm} (4–25)

4.4 Partial Learning Sequential GAN (P-GAN)

As described in earlier sections, current existing sequential GAN schemes are all based on entire sequential GAN. That is, GAN can only give scores/loss for an entire sequence when it has been generated or input [100]. There is no prior work on studying partially sequential GAN.
In this section, we will propose the first partial learning sequential GAN scheme which we call “Partial GAN” or P-GAN.

Based on the GAN framework described in previous sections, the proposed P-GAN framework consists of two adversarial modules, a generator $G$ and a discriminator $D$, which is illustrated in Fig. 4-4. The whole procedure includes feature learning and anomaly detection under such framework.

**4.4.1 Generative Adversarial Partial Learning**

We propose the idea of partial learning which involves the interaction between the partial state of generator and discriminator. This is fundamentally different from existing schemes.

Figure 4-4. Illustration of our proposed P-GAN training framework.
such as SeqGAN \cite{100} in which the generator blindly generates a whole sequence that the
discriminator relies upon to make decision.

In Section 4.3, we have seen how LSTM works. To briefly reiterate, the LSTM network
processes a sequence of input and target pairs \((x_1, y_1), \ldots, (x_m, y_m)\). For each pair \((x_i, y_i)\) the
LSTM network takes the new input \(x_i\) and produces an estimate for the target \(y_i\) given all the
previous inputs \(x_1, \ldots, x_i\). The past inputs \(x_1, \ldots, x_{i-1}\) determine the state of the network that
comprises a hidden vector \(h \in \mathbb{R}^d\) and a memory vector \(m \in \mathbb{R}^d\). The computation at each
step is defined as follows \cite{119}:

\[
g^u = \sigma(W^u H), \tag{4–26}
\]
\[
g^f = \sigma(W^f H), \tag{4–27}
\]
\[
g^o = \sigma(W^o H), \tag{4–28}
\]
\[
g^c = \tanh(W^c H), \tag{4–29}
\]
\[
m' = g^f \odot m + g^u \odot g^c, \tag{4–30}
\]
\[
h' = \tanh(g^o \odot m'), \tag{4–31}
\]

where \(\sigma\) is the logistic sigmoid function, \(W^u, W^f, W^o, W^c\) in \(\mathbb{R}^{d \times 2d}\) are the recurrent weight
matrices of the network and \(H \in \mathbb{R}^{2d}\) is the concatenation of the new input \(x_i\), transformed by
a projection matrix \(l\), and the previous hidden vector \(h\):

\[
H = \begin{bmatrix} lx_i \\ h \end{bmatrix}. \tag{4–32}
\]

The computation outputs new hidden and memory vectors \(h'\) and \(m'\) that comprise the next
state of the network. The estimate for the target is then computed in terms of the hidden
vector \(h'\). We use the functional \textsc{LSTM}(\ldots) as shorthand for Eq. \((4–26) – (4–31)\) as follows:

\[
(h', m') = \textsc{LSTM}(H, m, W), \tag{4–33}
\]

where \(W\) concatenates the four weight matrices \(W^u, W^f, W^o, W^c\).
In our proposed P-GAN (as illustrated in Fig. 4-4), the generator $G$ learns a distribution $p_g$ over data $x$ via a mapping $G(z)$ of samples $z$, 1D vectors of uniformly distributed input noise sampled from latent space $Z$, to sequential data space manifold $X$, which is populated by positive examples. Different from classical sequential GAN, during training the generator also receives the internal state information from discriminator in every LSTM step. That is, the LSTM in $G$ receives as input the hidden vector $h_d$ and memory vector $m_d$ from discriminator $D$.

The computation proceeds as follows. For a time step $t$, the generator of P-GAN model first concatenates the input hidden vector from discriminator as:

$$H_g = \begin{bmatrix} lzt \\ h_g \\ h_d \end{bmatrix},$$

(4–34)

where $z_t$ is the noise input at time step $t$, $h_g$ is the hidden state of generator $G$ and $h_d$ is the hidden state of discriminator $D$. Then the model computes transform $\text{LSTM}(\ldots)$, one for each $G$ and $D$, obtaining the desired output hidden and memory vectors:

$$(h'_d, m'_d) = \text{LSTM}(H_d, m_d, W_d),$$

(4–35)

$$(h'_g, m'_g) = \text{LSTM}(H_g, m_g, W_g).$$

(4–36)

Each transform takes discriminator and generator into computation and applies the LSTM mechanism to incorporate the hidden states from both. The memory vectors are also incorporated in the transform to represent the distinctive part of $G$ and $D$.

### 4.4.2 Time Step Extension

In some situations, we may find it useful for the model to incorporate more information from previous time steps. Fortunately, the proposed model in 4.4.1 can be easily extended to incorporate longer previous time steps.
To extend to longer time steps, the generator can utilize previous calculated hidden states to form a new vector $H'$ as follows:

$$H' = \begin{bmatrix} l_z_t \\ h_g \\ h'_d \end{bmatrix},$$  

(4–37)

where $h'_d$ is the hidden state vector from discriminator D that may come from arbitrary previous time steps. Then $H'$ can be used for final transformation to obtain the next hidden and memory state using Eq. (4–35) and (4–36).

### 4.4.3 Model Training

Traditionally, a discriminator only makes decision (classifies) after the whole sequence from the generator has been generated. In P-GAN, however, for any one time step $t$, the output vector $v_t$ obtained from the LSTM units in discriminator can be used for sequence identification:

$$y_t = \text{softmax}(W_cv_t + b_c),$$  

(4–38)

where $y_t$ is a $K$-dimensional vector $\{p_1, p_2, \ldots, p_k\}$. The semi-supervised learning with multi-class identification techniques can also be utilized here.

Assuming half of our data set consists of real data and half of it is generated (this is arbitrary), our loss function at time step $t$ for training the classifier then becomes

$$L_1^t = -E_{x,y\sim p_{data}(x,y)}(\log p_{model}(y_t|x, y_t < K + 1)), \quad (4–39)$$

$$L_2^t = -E_{x\sim p_{data}(x)}(\log 1 - p_{model}(y_t = K + 1|x)) - E_{x\sim G}(\log p_{model}(y_t = K + 1|x)), \quad (4–40)$$

where we have decomposed the total cross-entropy loss into our standard supervised loss function $L_1$ (the negative log probability of the label, given that the data is real) and an unsupervised loss $L_2$ which is in fact the standard GAN game-value as becomes evident when we substitute $D(x) = 1 - p_{model}(y_t = K + 1|x)$ into the expression:

$$L_2^t = -E_{x\sim p_{data}(x)}(\log D(x_t)) - E_{x\sim \text{noise}}(\log 1 - D(G(z_t))). \quad (4–41)$$
The target loss function we are trying to minimize becomes:

\[
L_1 = E \left\{ \sum_{t=0}^{T} \alpha_t L_1^t \right\}, \quad (4-42)
\]

\[
L_2 = E \left\{ \sum_{t=0}^{T} \alpha_t L_2^t \right\}, \quad (4-43)
\]

where \( \alpha_t \) is the weight for the corresponding loss at time step \( t \). Then by minimizing both \( L_1 \) and \( L_2 \) iteratively, we are able to conduct the training process.

### 4.4.4 Model Building

In P-GAN setting, the network architecture of the generator \( G \) is equivalent to a sequential decoder that utilizes a sequence of stacked LSTM. The discriminator \( D \) is a sequential RNN that maps sequential data to a single scalar value \( D(\cdot) \). The discriminator output \( D(\cdot) \) can be interpreted as probability that the given input to the discriminator \( D \), was a normal sequence of \( x \) sampled from training data set \( \mathcal{X} \) or generated \( G(z) \) by the generator \( G \). \( D \) and \( G \) are simultaneously optimized through the following two-player min-max game with value function \( V(G, D) \):

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim \rho_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim \rho_G(z)} [\log(1 - D(G(z)))] . \quad (4-44)
\]

Algorithm 4.1 shows the process to build a generator corresponding to Fig. 4-4.

**Algorithm 4.1. Create Generator**

1. **procedure** BuildGenerator(\( z \)) \( \triangleright \) \( z \) is input vector \( (z^{(0)}, z^{(1)}, \ldots, z^{(t)}) \) as in Fig. 4-4
2. \( n_h = \text{number of LSTM unit} \)
3. \( \text{lstm} = \text{create LSTM cell with} n_h \text{ unit} \)
4. stack \( \text{lstm} \) to create deep structure. \( \triangleright \text{build deep LSTM} \)
5. \( \text{output}[i] = \text{run LSTM with} z \text{ for each step} i. \) \( \triangleright \text{store to output}[i] \)
6. build dense network with \( h_d \) hidden layers.
7. \( x_g[i] = \text{input output}[i] \text{ to dense layer.} \) \( \triangleright x_g \) is the generated output from dense layer.
8. return \( x_g \) \( \triangleright x_g \) will be input to discriminator.
9. **end procedure**

The discriminator is trained to maximize the probability of assigning real training examples the “real” and samples from \( \rho_g \) the “fake” label. The generator \( G \) is simultaneously trained to
fool D via minimizing $V(G) = \log(1 - D(G(z)))$, which is equivalent to minimizing

$$V(G) = -D(G(z)). \quad (4-45)$$

During adversarial training the generator improves in generating realistic sequence data while the discriminator progresses in correctly identifying real and generated sequence data.

Algorithm 4.2 illustrates the basic structure and process to build the discriminator.

**Algorithm 4.2. Create Discriminator**

1: procedure BuildDiscriminator($x, x_g$) \hspace{1em} \triangleright $x$ is input vector $(x^{(0)}, x^{(1)}, ..., x^{(t)})$ as in Fig. 4-4
2: \hspace{1em} $x_a = \text{combine}[x, x_g]$ \hspace{1em} \triangleright concatenate $x$ and $x_g$ as input.
3: \hspace{1em} lstm = create LSTM cell with $n_h$ unit.
4: \hspace{1em} stack lstm to create deep structure. \hspace{1em} \triangleright build deep LSTM.
5: \hspace{1em} output = run LSTM with $x_a$ for sequence from 0 to $t$. \hspace{1em} \triangleright get the result from timestamp $t$.
6: \hspace{1em} $d = \text{build dense network with } h_d \text{ hidden layers}.$
7: \hspace{1em} result = input output to dense layer $d$.
8: add sigmoid activation function on top of dense layer.
9: split result into $y_d$ and $y_g$ \hspace{1em} \triangleright $y_d$ is output for input $x$ and $y_g$ the output for input $x_g$.
10: return $y_d$ and $y_g$ \hspace{1em} \triangleright the return values will be used in cost functions.
11: end procedure

Having built generator and discriminator, the next step is to train the GAN. The adversarial training procedure is shown in Algorithm 4.3.

**Algorithm 4.3. Adversarial Training**

1: procedure AdversarialTraining
2: load sequential data.
3: $x_g = \text{build generator by calling Algorithm 4.1.}$
4: $y_d, y_g = \text{build discriminator by calling Algorithm 4.2.}$
5: loss$_d = -E \{ (\log y_d^\prime + \log(1 - y_g^\prime)) \} \hspace{1em} \triangleright$ define loss$_d$ as loss function for discriminator as in eq. (4-42)
6: loss$_g = -E \{ \log y_g^\prime \} \hspace{1em} \triangleright$ define loss$_g$ as loss function for generator as in eq. (4-43).
7: optimizer$_d = \text{Adam optimizer of loss$_d$} \hspace{1em} \triangleright$ define Adam optimizer
8: optimizer$_g = \text{Adam optimizer of loss$_g$}$
9: for each training epoch do
10: \hspace{1em} train the discriminator by optimizing optimizer$_d$
11: \hspace{1em} train the generator by optimizing optimizer$_g$
12: end for
13: end procedure
It usually takes a few dozen epochs to complete the training process.

4.5 Experiment Results

In this section, we will conduct experiment for our proposed P-GAN framework.

4.5.1 Experiment Setup

As described in Chapter 3, the user profile dataset we collected are complex, high-dimensional and are challenging and time-consuming to be processed. Therefore, before we can do anything with the dataset, first and foremost dimension reduction steps are inevitable. This steps are the same with the description in Section 3.3.

After dimension reduction, the dataset are split into two groups, one for training and one for testing. The dataset selection and pre-processing steps also remain the same as described in Section 3.5. Please refer to these relevant chapters and sections for details for steps of preparing data. The training framework involves a generator and a discriminator and they both train iteratively. We build the system using Tensorflow [99]. The both generator and discriminator are consisted of a deep LSTM layer and a dense layer for output. Both of their LSTM layers consist of 300 LSTM units.

Greatly different from AttenGAN setup as described in Section 3.5.4, P-GAN does not require a fixed sequence length $T$. Conversely, the length of the sequence can be dynamically determined during the training procedure.

4.5.2 User Profile Identification Accuracy

User profile identification accuracy measures how accurate the model can make identification by testing it on a given dataset. It has relation with number of correctly predicted class with total number of samples. The identification accuracy $p$ given a model is calculated by:

$$p = \frac{\text{Number of correctly predicted class}}{\text{total number of samples}}. \quad (4–46)$$

Fig. 4-5 shows the relation between identification accuracy and number of epochs. There are 4 schemes that are in comparison which include our proposed P-GAN, a type of sequential GAN (SeqGAN) [100], a simple LSTM classifier and a classical deep neural network (DNN).
classifier. Note that the discriminator of SeqGAN is a modified version that can output classes instead of simple 0/1 output.

From these plots, we are able to make the following observations. First, the accuracy measurement of all of these approaches increases during training. That is, as the number of epochs increases, the accuracy increases accordingly. Second, our proposed P-GAN achieves much better accuracy performance than SeqGAN, plain LSTM and DNN. DNN achieves the worst accuracy performance. This fact can be further justified by taking a look at the specific accuracy value at epoch 30, shown in Table 4-1. The reason is obvious, as both SeqGAN and LSTM are not able to capture the partial sequence information and hence cannot make
Table 4-1. P-GAN identification accuracy comparison at Epoch 30.

<table>
<thead>
<tr>
<th></th>
<th>P-GAN</th>
<th>AttenGAN</th>
<th>SeqGAN</th>
<th>LSTM</th>
<th>DNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.978</td>
<td>0.988</td>
<td>0.941</td>
<td>0.911</td>
<td>0.867</td>
</tr>
</tbody>
</table>

decisions in the middle of sequence. To make things worse, DNN even cannot capture the time series information. We put it here simply for comparison purposes.

We also notice that the performance of identification accuracy of P-GAN is comparable with AttenGAN proposed in Chapter 3 if not better than that, and the actual identification accuracy of P-GAN at epoch 30 is slightly lower than AttenGAN in Table 4-1. However, P-GAN is more advantageous in that it does not require a fixed sequence length during learning. This feature can significantly reduce feedback delay from discriminator to generator from $T$ time slots to 1 time slot. It also greatly improves the adaptivity of system to dynamic time series where a fixed time step $T$ is usually difficult to determine optimally in real-life complex dataset.

Third, we can see that the three GAN based scheme – P-GAN, AttenGAN and SeqGAN – generally perform better than classical neural network schemes – LSTM and DNN. This further justifies our hypothesis that GAN has the potential to build strong and better classifiers as well.

### 4.5.3 Anomaly Detection Capability

Anomaly detection capability is often measured in receiver operating characteristic (ROC) curve. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. True positive rate measures the proportion of positives that are correctly identified as such and false positive rate is the probability of falsely rejecting the null hypothesis for a particular test. The false positive rate is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events (regardless of identification, i.e., anomalies).
We plot the ROC curve for our 4 comparison schemes: P-GAN, AttenGAN, SeqGAN and LSTM. Again, the SeqGAN is specifically made modifications on the output to adapt to identification tasks. The results are shown in Fig. 4-6.

![ROC Curves](image)

Figure 4-6. P-GAN ROC curve compared with other schemes.

From these plots, we are able to make the following observations. First, the TPR measurement of all of these approaches increases accordingly as the FPR increases. Second, thanks to the partial learning mechanism, our proposed P-GAN is able to achieve the best TPR performance among all at given FPR value, followed by AttenGAN proposed in Chapter 3. The reason behind is due to partial learning which enables the generator to learn from discriminator at every time step. This timely feedback mechanism ensures that P-GAN will be more sensitive to anomaly sequence, thus providing better overall performance in detecting anomaly while
keeping FPR low outperforming AttenGAN. SeqGAN and LSTM performs poorly as expected since they are not able to capture the partial learning information in our sequence. Third, we can see that the three GAN based schemes – P-GAN, AttenGAN and SeqGAN – generally perform better than classical LSTM scheme. This further justifies our hypothesis that GAN has the potential to build strong and better anomaly detector as well.

4.5.4 Convergence Speed

Another interesting fact we are interested in is how fast the learning algorithm learns. This can be measured by the non-negative log likelihood (NLL) after a number of epochs. We call this convergence speed. The NLL is calculated by:

$$NLL = -\mathbb{E} \left\{ \sum_d \log p_d \right\},$$

(4–47)

where $j$ is the label of sequence $d$ and $p$ is the probability that $d$ is predicted to belong to label $j$.

As illustrated in Fig. 4-7, we plot the curves that represent the relations between number of epochs for training and the NLL loss value.

From these plots, we are able to make the following observations. First, all the schemes are converging fast in the first few epochs (less than 5) of training. This is because all the internal states were randomly set when the system initializes. In the first few epochs, any action of learning will drive the system into convergence rapidly. After that, the system’s learning speed would slow down and finally reach a stable state at around epoch 15. This is an expected behavior. Second, thanks to the partial learning mechanism, our proposed P-GAN is able to achieve the fastest convergence speed among all (blue line), followed by AttenGAN, SeqGAN and LSTM. This is due to the partial learning mechanism in P-GAN where it helps P-GAN converge fast even starting from the first few epochs. P-GAN can always keep NLL value low until epoch 30. Furthermore, the three GAN based schemes perform much better than LSTM and DNN. This further justifies our hypothesis that GAN has the potential to build strong and better anomaly detector as well. Third, DNN performs worst. This is a good old
approach which does not have too much performance advantage when compared with more advanced GAN based approaches.

4.6 Chapter Summary

In this chapter, we have proposed an anomaly detection framework based on partial generative adversarial networks called “Partial GAN” or P-GAN. To the best of our knowledge, this is the first work that is based on partial sequence learning of GAN to perform user profile identification and anomaly detection for sequential user data. The proposed scheme P-GAN works very well with our dataset and the results also shed light on that GAN is a good candidate and has potential to become a very effective framework for sequential data identification and anomaly detection.
CHAPTER 5
CONCLUSIONS

This dissertation was about the research of building a new framework for intelligent network and the pertaining security problems. We approached the research within both network domain and machine learning domain, and extensive technique details were covered in the chapters.

Although there are much more topics and works I had covered and accomplished during my years of PhD study, we presented the select research in 3 major topics:

1) In the first topic (reported in Chapter 2) we presented a practical distributed scheduling framework for QoS-aware small cell mmwave mesh backhaul network.

We have considered the problem of backhaul scheduling to maximize the system throughput while having flows’ QoS requirements satisfied in the mmWave backhaul network. The formulated problem was NP-hard, thus we proposed the DMQ scheduling algorithm, where it was designed to achieve the spatial reuse while reducing interference and channel contention between neighbors in a coarse-to-fine two-phase fashion.

The QoS aware priority was further exploited in the algorithm to provide better QoS guarantees for flows. We conducted extensive experiments and results have shown that our proposed algorithm is able to achieve much higher system throughput and much lower packet loss ratio than other existing schemes under different criteria.

2) In the second topic (reported in Chapter 3) we proposed a detection framework for user profile anomalies. We presented the complexity of the dataset we were collecting and challenges we were facing.

First, we described the format of our dataset and the procedure we had taken to convert them to machine learning compatible format. This included hashing and proper pre-processing techniques employed.

Next, we explained the reason that dimension reduction was necessary due to the volume size of our dataset. We compared multiple dimension reduction techniques that were widely
used nowadays and concluded that a stacked autoencoder based scheme best fitted our use case. We then described in detail the idea of autoencoder and the stacked autoencoder and how we employed them into our system.

Lastly, we proposed a brand new GAN framework – AttenGAN for our classification and anomaly detection tasks. We described how AttenGAN works for our tasks and how we designed and implemented our system. The experimental results showed the superior of our proposed framework.

3) In the third topic (reported in Chapter 4) we presented a more advanced approach for anomaly detection for user profile dataset as we sought to partial Generative adversarial models.

We first gave an introduction to why the problem is worth studying. Next, since our proposed scheme was based on LSTM, we gave a brief review of LSTM as well. However, traditional applications of GAN were all focused on image generation and related image field.

Based on sequential GAN, we then proposed a framework for detecting sequential anomaly, that we have formally proposed our P-GAN framework and explained how it works. The proposed framework was composed of a GAN for partial feature learning and a weighted sum loss function for classification and anomaly detection tasks. The experiment results showed that our proposed P-GAN framework was able to efficiently detect anomaly and the accuracy was high.
REFERENCES


BIOGRAPHICAL SKETCH

Jiade Li received the B.E. degree in automation from the University of Science and Technology of China in 2010, and the M.S. degree in optics from the University of Chinese Academy of Sciences in 2013. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Florida and is expected to graduate in 2017. Jiade worked as a Ph.D. software engineer intern for Facebook in Summer, 2017. His research interests include machine learning, computer vision, network congestion control and network security.