CHALMERS
UNIVERSITY OF TECHNOLOGY

Deep Learning Methods Towards Infotainment Systems

Study of a sequential generative adversarial network and LSTM-based system

Master's thesis in System Control and Mechatronics

Min Cheng

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Deep Learning Methods Towards Infotainment Systems

Study of a sequential generative adversarial network and LSTM-based system

Min Cheng

Supervisor: Ken Ryrbo in CEVT
Prof Irene Y.H. Gu (Chalmers University of Technology)

Examiner: Prof Irene Y.H. Gu

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Deep Learning Methods Towards Infotainment Systems

Study of a sequential generative adversarial network and LSTM-based system

Min Cheng

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX[1]
Gothenburg, Sweden 2019

Abstract

Automotive vehicles are sophisticated systems embedding with many subsystems such as infotainment, power-train, exterior light, etc. CEVT has been developing automated testing using auto-generated python scripts to execute various functions on expensive equipment to investigate these systems' behavior. It is time-consuming to execute test scripts. In this thesis we investigate and develop an RNN based log-file prediction method in order to improve the automated testing efficiency. Furthermore, a generative adversarial model for data augmentation has been investigated to verify whether the prediction performance can be further improved. The results show that the LSTM detection system can achieve around 86% accuracy on a small data-set and it can save up to 60% of execution time to improve the test efficiency.

Keywords: infotainment system, deep learning, LSTM, generative adversarial network, RNN, LeakGAN.

Acknowledgements

First, I would like to thank my supervisor Ken Ryrbo from CEVT and my supervisor and examiner Irene Yu-Hua Gu at Chalmers University of Technology for giving me this opportunity to conduct this thesis project, and also thank PhD student ChenJie Ge for co-supervising, consistency support and valuable suggestions during the research.

Min Cheng, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Modern automotive vehicles are sophisticated systems embedding with many subsystems such as infotainment, power-train, exterior light, etc. These subsystems are controlled by Electronic Control Unit (ECU) which is a embedded system to provide better functionality incorporation of different module and improve the efficiency. Since ECU has much functionality and therefore, it is needed to be tested automatically to guarantee that both software and hardware works properly. A fault in system can hardly risk people's live, but it will still impact the user's experience. Together with CEVT AB, the division of EE(electronic engineering) is currently exploring a new infotainment prototype for the next-generation system which is an android-based under the hood. With an increasing of systems complexity, Manual testing is time-consuming, tedious and requires substantial investment in human resources[4].

A new approach that is being tried at CEVT is to use system models that describe the possible behavior of the system under test in order to auto-generate python scripts that execute these functions in real time, either in vehicle, or at an earlier stage in production, Hardware In the Loop setups, which are super computers designed to simulate all or part of the vehicle system. These test executions result in log files with details about the test execution such as executed functions and information about whether the test case passed or failed. Since the test scripts run in real time on expensive equipment, test execution time is an essential factor that CEVT wants to reduce. Since the generation of test scripts is rapid, but the execution is rather slow, there is a need to try to increase the number of failed test cases per unit of time, so that more bugs can be found earlier in production, reducing overall cost. CEVT would therefore like to investigate the possibility of first predicting which test cases will fail, based on prior experience, and then sorting out these for execution and hopefully increasing the proportion of failed test cases to passed test cases during real time test execution. The method to investigate and implement for this thesis is a machine learning approach in order to try to classify test scripts as pass/fail based on their input and thus enabling filtering out auto-generated test scripts that have a very low probability of resulting in a failed execution

1.2 Purpose

For the purpose of this project is to improve the efficiency for automated testing, log files will be provided for analysis. These log files contains detailed information about each executed automated test case, including input data and output data. Up to at least 15000 of log-files has been produced and collected by CEVT and provide to this thesis project. The goal of this project is to investigate and develop a suitable deep learning methods based on the log-file. The log-files provided contains the input and output of the automated test execution as data concerning executed steps, timestamps and pass/fail/error outcome of each action.

The research topic can be varied from simple to very complex. The most simple way is to perform supervised learning through label pass and fail test cases separately to examine whether the incoming test cases can be predicted correctly. This helps to facilitate better awareness of this specific data as a proof of concept.

1.3 Objective

To achieve the purpose, the project is divided into following parts:

- To find suitable input vector for deep learning by selecting some relevant components from high dimensional log-file. This is largely done through empirical tests. Different approaches with diverse parameters will be tried and compared with each other for the highest possible accuracy in prediction.
- To seek suitable deep learning methods to predict the outcome of a small sample Body Domain system to achieve the highest prediction accuracy.
- To examine whether data augmentation for log-files is achievable by applying Generative Adversarial Networks.

1.4 Scope

During the data collection, the output of log files error output has different formats and outlines, thus it is challenging to write a specific program for general error collection without containing any unwanted noise. It is a trade-off between large data extraction with low quality and lesser data extraction with high accuracy. The project is mainly focusing on supervised learning.

1.4.1 Limitation

When finishes the thesis project, the program/script has not been performed on the whole infotainment system due to the unexpected software delay in the company and lack of data set. Fortunately, another department in Pannverkstad use similar python scrip for testing vehicle's Body Domain system such as Exterior light part and produce same format of log-file. Therefore, the main scope of this project has been changed from infotainment system to the Body Domain system for the exterior light part. But the thesis work has proved in the sample of body domain system that the whole concept is valid and effective, and this approach will not only be employed in the exterior light system, but also in the other hardware/software systems like infotainment later to improve the test efficiency.

2
Theory and Method

This chapter describes some basic concepts in machine learning, the fundamental theory and methods with related work of various models which have been studied and applied in this project.

2.1 Machine Learning Methods

Machine learning(ML) [5] is a multidisciplinary sub-field within the area of computer science which covers a series of subjects e.g., statistics, probability theory, etc. In the last decade with the increase in available computational power, machine learning algorithms have been developed vastly and widely applied in various applications such as image processing, natural language, self-driving vehicle, etc. Machine learning algorithms allow the computer to learn from preexisting data-sets and make a prediction based on early learning experience, which means ML has the ability to handle the data without being explicitly programmed.

2.1.1 Deep Artificial Neural Networks

Human brain neurons [6] with synapses and terminals to receive and transmit information from our sense, with many can build up a network which can handle different complex information. Artificial Neural is a type of method designed for mimicking the human brain with a similar concept. With innumerable artificial neural working together, it is possible to build up an Artificial Neural Network to represent our human brain's interaction.

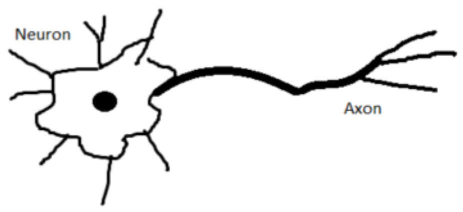


Figure 2.1: Figure illustrates the single neuron of human brain

2.1.2 Classification Problem

Classification problem is a central topic in machine learning which refers to identifying a set of dataset to categorize which class it belongs to. For instance, in IMDb movie review, researchers can use machine learning algorithms to train neural networks with a set of dataset which consists of different review sentences that have been labeled as negative or positive. The neural network can utilize training data to learn the underlying characteristics of negative/positive comment so that it can perform classification prediction when it encounters new datasets.

2.2 Neural Network

2.2.1 Feed forward Neural Network

The basic unit of a neural network is a single neuron Figure 2.2 illustrates the most simple architecture of a feed-forward neural network that can be constructed.

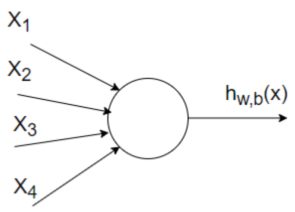


Figure 2.2: Illustration depicting the principle of single neuron and corresponding weight bias and the activation function.

Taking supervised learning as an example. Suppose a training sample : input vector $x \in \mathbb{R}^n$, weight vector $w \in \mathbb{R}^n$, and threshold $b \in \mathbb{R}^n$, then the output y is:

$$y = f(w^T x + b) = f(\sum_{i=1}^{n} w_i x_i + b) \tag{2.1}$$

Each input is multiplied with its weight respectively and summed together. The activation function can be e.g. either Rectified Linear Units (ReLU) or a sigmoid function. A linear model can only approximate linear relations between the input and output. Thus it is crucial to apply a non-linear activation function f since it allows the neural network to represent non-linear dependencies. With many such single neurons, we can build up a simple feed-forward neural network.

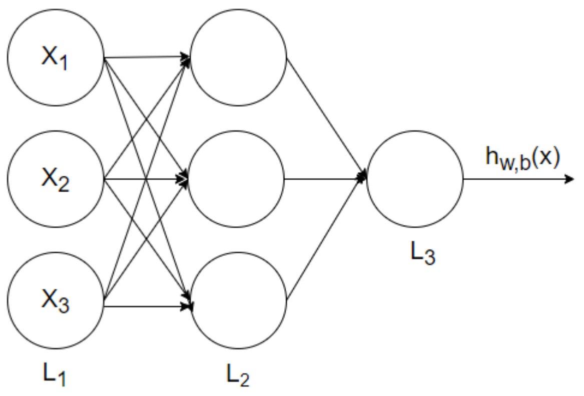


Figure 2.3: Illustration depicting the principle of a simple feed-forward neural network.

The illustration above illustrates a simple neural network. The first layer is called the input layer, the last layer is the output layer, the layer L_2 noted is called a hidden layer because no node in the middle layer usually cannot be observed.

2.2.2 Recurrent Neural Network

A Recurrent Neural Network (RNN) [7] is a type of neural network which allows neurons to receive states from a previous time, with this characteristic. RNN has the ability to memorize previous states like the human brain and hence, RNN is suitable to handle sequential data. For instance, normal feed machine translation or voice recognition. Figure 2.4 shows the structure of RNN, the recurrent neural

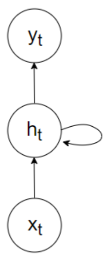


Figure 2.4: Recurrent Neural Network

network can be expressed by following formulas, where h_t is a hidden state at time step t, W is the weight matrix, f is the activation function, O_t is the output at time step t, b and c is the bias for hidden layer and output layer accordingly:

$$a_t = b + Wh_{t-1} + Ux_t \tag{2.2}$$

$$h_t = f(a_t) \tag{2.3}$$

$$o_t = c + Vh_t \tag{2.4}$$

At every time step t the input is parameterized by a weight matrix U, h_t is denoted as a hidden layer, every step previous hidden layer is multiplied by a weight matrix

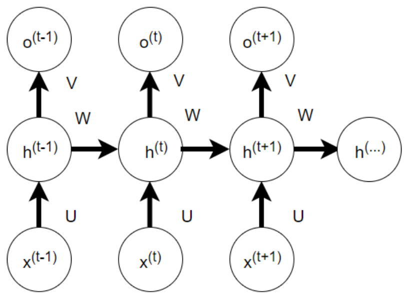


Figure 2.5: Time-unfolded computational graph

W feed-forward to the next step hidden layer, the connection between hidden layer and output is parameterized by matrix V [7].

In summary, with feedback loops, the recurrent neural network can keep not only the previous time step information but also contains all historical information in reserve implicitly in the hidden unit and transmit data in next time-step. This feature allows RNN to perform sequence data which a Feed-forward network can not.
As similar as feed-forward network, vanishing gradient descent which described in section 2.3.1 is a major barrier for RNN to handle the long time sequence formation. The conventional RNN updates the weight and bias via backpropagation (BPTT).

2.2.2.1 Long Short-Term Memory

Long short-term memory is a specific type of RNN that was proposed by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [8] aims to solve the vanishing gradient descent problem.

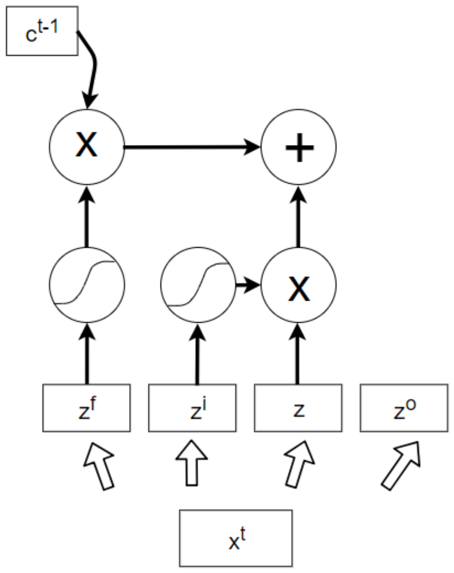


Figure 2.6: Figures shows the internal state algorithm.

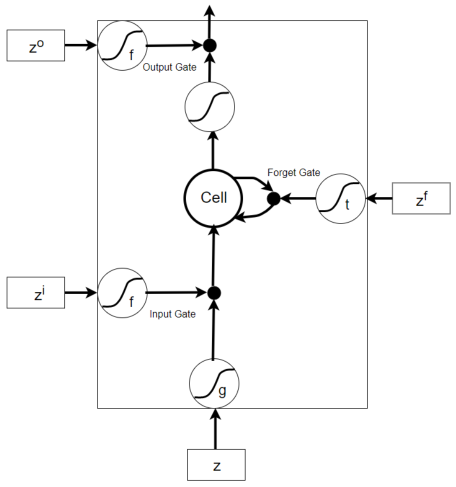


Figure 2.7: Illustration depicting the internal structure of LSTM neurons.

Figures 2.6 and 2.7 illustrate the example of structure of LSTM, each cell has three gates: input gates I, forget gates F and output gates O. By updating the weight through Backpropagation Through Time, where Z^f,Z^i,Z,Z^o are the weight corresponding with different gates and all depending on all previous hidden states and current input, then times with different weights accordingly.

$$Z = \tanh(W[x^t, h^{t-1}]) \tag{2.5}$$
$$Z^i = \sigma(W^i[x^t, h^{t-1}] \tag{2.6}$$
$$Z^f = \sigma(W^f[x^t, h^{t-1}] \tag{2.7}$$
$$Z^o = \sigma(W^o[x^t, h^{t-1}] \tag{2.8}$$

These weights learn automatically whether to forget or remember previous information. Compared to RNN which only has one transfer state h^t , LSTM has two transfer states, cell state c^t , hidden state h^t, four inputs, and one output. The current cell state equals to the forget gate's weight times previous cell states and sum with input gates weight time input weight. The forget gates which control the cell states base on previous hidden layer state h_{t-1} and input state x_t

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t]) \tag{2.9}$$

With the combination of previous hidden state and input, multiplied with the input gates weight, and passed through a sigmoid function, obtaining the input gate i_t with a value between 0 and 1, which is the percentage decided how much information is allowed to pass through from the input. In order to update the cell state C_t, the previous cell state C_{t-1} scale by forget gate f_t and summing with $\tilde{C}$ times with input gate i_t, which allows Lstm separately decide when to forget old information or when to obtain new information. [2], where the $\tilde{C}$ is the candidate state value between zero and one depends on the previously hidden layer and current input.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t]) \tag{2.10}$$
$$\tilde{C} = \tanh(W_c \cdot [h_{t-1}, x_t]) \tag{2.11}$$
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{2.12}$$

The value of output gates are derived by output of sigmoid functions which include previous hidden state and current input state, are multiplied by output weight.
Lastly, the current hidden state is derived by the multiplication of the output gate o_t and a hyperbolic tangent function which is fed with current cell state C_t

$$o_t = \sigma(W_o[h_{t-1}, x_t]) \tag{2.13}$$
$$h_t = o_t * tanh(C_t) \tag{2.14}$$

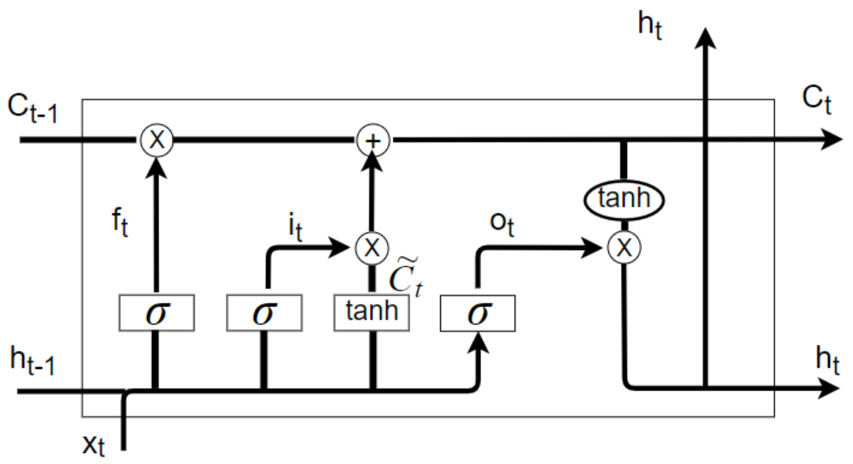


Figure 2.8: Illustration depicting the internal structure of a Lstm neuron [2]

2.2.3 Convolutional Neural Network

Convolution Neural Network(CNN), originally invented for computer vision, is one of widely used architecture can attain top performance in image processing [9]. A new manner was proposed by Kim [10] has shown that with little hyperparameter tuning, CNN can also achieve great effectiveness in sequential data classification problem. In order to get dense representation for sequence information. An example of such method is Word Embedding which has been mentioned in Section 2.2.5.3. CNN applies a specific receptive field called 1D convolution since it convolves only in one direction. With such convolutional filter, it can extract n-gram feature (described in section 2.2.5.3) from the sentence with various of filters kernel size which has been shown in the Figure 2.9

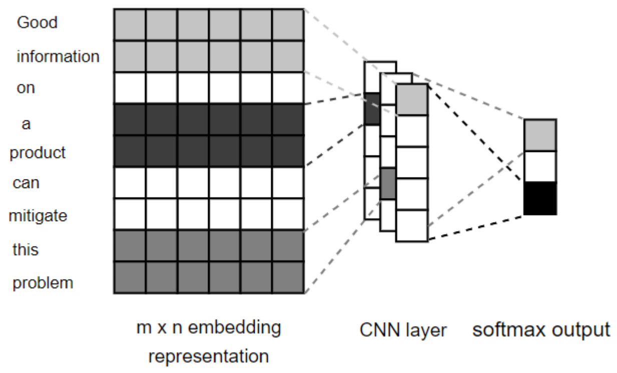


Figure 2.9: N-gram feature extraction by using 1D-convolutional filter

A max-pooling operation is conducted after convolutional layer in order to further condense the feature information. Max-pooling is a down-sampling strategy where the only maximum value of each subset is retained to represents the subset feature from previous layer. It reduces the computational cost and can prevent the overfitting problem.

2.2.4 Generative Adversarial Network

Generative Adversarial Network(GAN) is a type of generative model first proposed by (Goodfellow et al.,2014) [11], unlikely discriminative model used in statistical classification in supervised machine learning. Typically, a generative model such as GAN which is capable of learning data distribution $P_{data}(x)$ from latent space by given sample and generating realistic synthetic data, i.e. vivid portrait. GAN usually is composed of two sub-model, a generator (G) and a discriminator (D). The basic idea of GAN is to maximize the likelihood of generator parameter θ_G by given a true data distribution $P_{data}(x)$. Such maximum likelihood estimation can be described as following formula:

$$\theta_G^{opt} = \arg\max_{\theta} \prod_{k=1}^{z} P_G(x^z; \theta_G) \tag{2.15}$$

Where x^z are samples from the true data distribution $P_{data}(x)$. Multipling with log and turn a log of products into a sum of logs derives:

$$\arg\max_{\theta} \log \prod_{k=1}^{z} P_G(x^z; \theta_G) = \arg\max_{\theta} \sum_{k=1}^{z} \log P_G(x^z; \theta_G) \tag{2.16}$$

This can be rewritten as expectation value of likelihood from all sample X.

$$\theta_G^{opt} \approx \mathbb{E}_{x \sim p_{data}}[\log P_G(x; \theta)] \tag{2.17}$$

However, it is hard to maximize the likelihood θ_G^{opt} without given any feedback to generator and hence, a discriminator is introduced in order to evaluate the difference between true data distribution $P_{data}(x)$ and synthetic data distribution $P_G(\theta_G)$ which produced from generator. These two models compete with each other through adversarial training.
During the training process, in each iteration, the generator mimics the true data distribution and update it's parameter through back-propagation and tries to fool the discriminator with more realistic input data, while the discriminator is trained with both labeled true data and synthetic data which generated from generator and is obviously a binary classification model. Such alternating training process endures until the generator converges to true data distribution. The following equation describes the loss function of GAN. In the training scheme, an interleaved training is conducted, the discriminator tries to maximize the expectation value of loss function in one step. Simultaneously, the generator attempts to minimize the expectation value in another step.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{x \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.18}$$

In each step, the discriminator parameter θ_d is updated through stochastic gradient ascent

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(x^i) + \log(1 - D(G(z^i)))] \tag{2.19}$$

And the generator parameter θ_G is updated through gradient descent, but fix the discriminator.

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^i))) \tag{2.20}$$

The original GAN has many issues during training, and it is well-known as proverbially hard to train, one main problem is model collapse this leads generator lose diversity and generated limited type of output. Another main issue is vanishing gradients due to the discriminator is ineffective and failed from distinguishing between fake and real input.

2.2.5 Generative Adversarial Network for Sequential Data Augmentation

2.2.5.1 SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient

The GAN which is proposed by (Goodfellow et al.,2014) [11] is designed for generating continuous data. It has two main problems impedes its application prospect in NLP. For instance, in image processing, the generator can be updated by slightly changing model parameters in the direction of gradient of the loss from discriminator in pixel scale in order to generate a more realistic image. Nevertheless, it is inapplicable in NLP, since word tokenization makes sentence presentation in high dimension space is not continuous and cannot find the corresponding sentence with a slight change of parameter and hence, the loss from discriminator is not differentiable [12]. Secondly, the reward from discriminator can be only obtained with an entire sequence has been generated. This makes the 'guiding signal' sparsely and hardly to judge how good the partially generated sentence is [12].
In order to solve these two problems, a new framework SeqGan proposed by [12] has proved with promising result. With combining of Reinforcement learning ,by treating generator as a stochastic policy to make it feasible to update parameter through policy gradient [12].
Given a generator G_θ with parameter θ to produce a sequence $Y_{1:T} = (y_1, ...y_t, ...y_T), y_t \in \Gamma$, where Γ is volcabulary of available tokens, t is time-step. The stochastic generating policy for the next token denoted as $G_\theta(y_t|Y_{1:t-1})$. To update generator via Policy Gradient. We need to maximize the expected end reward as follows [12].

$$J(\theta) = \mathbb{E}[R_t|s_0, \theta] = \sum_{y_1 \in \Gamma} G_\theta(y_1|s_0) \cdot Q_{D_\phi}^{G_\theta}(s_0, y_1) \tag{2.21}$$

where R is a reward, R_T is the reward in final time T, $Q(s, a)$ is the action value with state s and action a, The expectation of the reward at final time T is the summation over all the Q values followed by policy G_θ [12]. In order to approximate the action-value function $Q(s, a)$, it is obtained by the discriminator as follows:

$$Q_{D_\phi}^{G_\theta}(a = y_T, s = Y_{1:T-1}) = D_\phi(Y_{1:T}) \tag{2.22}$$

The reward can only be returned when the sequence is fully generated, With such sparse reward, it is hardly to obtain instantaneous feedback for the model and difficult to inspect how good the currently produced token is. Therefore, in order to attain an intermediate reward in each time step, a Monte Carlo (MC) search with a roll-out policy G_β is introduced as [12], where the superscripts N is the roll-out number:

$$\{Y_{1:T}^1, ..., Y_{1:T}^N\} = \mathrm{MC}^{G_\beta}(Y_{1:t}; N) \tag{2.23}$$

Equation (2.24) describes the action value function in two cases, when $t = T$ in final time, it can get an intermediate rewards from discriminator, when sequence is partially produced for $t < T$, it adapts MC approach to produce complete sequence, starting rolling out from time t to the end, summarizes all the roll-out reward and average them as action-value:

$$Q^{G_\theta}_{D_\phi}(a = y_t, s = Y_{1:T-1}) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y^n_{1:T}), Y^n_{1:T} \in \mathrm{MC}^{G_\beta}(Y_{1:t}; N), & \text{for} \quad t < T \\ D_\phi(Y_{1:t}), & \text{for} \quad t = T \end{cases} \tag{2.24}$$

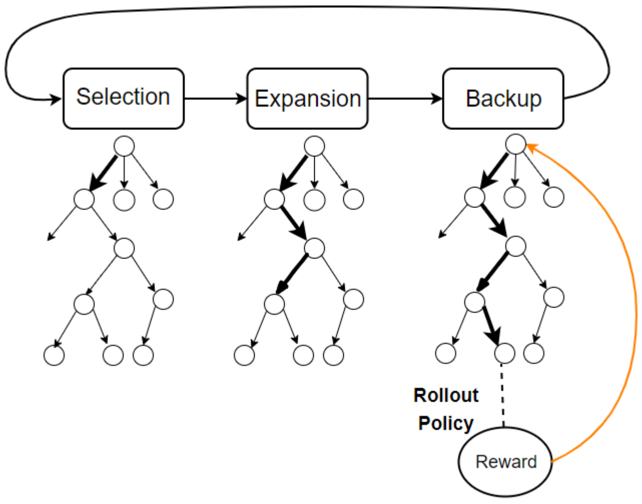


Figure 2.10: Figure illustrates the principle of Monte Carlo tree search.

The generator's parameters θ can be updated when a new discriminator is retrained

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{Y_{1:t-1}\sim G_\theta}[\sum_{y_t\in\Gamma} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) \cdot Q^{G_\theta}_{D_\phi}(Y_{1:t-1}, y_t)] \tag{2.25}$$

$$\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta) \tag{2.26}$$

Algorithm 1 illustrates the pseudocode of SeqGAN

Algorithm 1: SeqGAN algorithm

```
Initialize G(generator) and D(discriminator) with random weight ;
Pre-train G for N iterations using MLE with real data S ;
Using current G generate negative sample ;
Pre-train D with cross entropy ;
while SeqGAN not converges do
    for generator do
        for $t$ in 1:$T$ do
            Calculate reward Q with Eq 2.24
        Update generator parameter via policy gradient by Eq 2.25 and 2.41
    for discriminator do
        Train discriminator D for J iterations with Eq 2.18
```

2.2.5.2 LeakGAN: Long Text Generation via Adversarial Training with Leaked Information

LeakGan is new framework proposed by [3] aims to solve long sequence generation issue, similar to SeqGan, it adopts Policy Gradient to update the generator parameter. Although SeqGAN has a breakthrough for text generation by adapting a Policy gradient method, the reward can only be observed after the a complete sequence is generated. This makes the feedback lack of intermediate information of sequence structure even with MC-approach. The LeakGan is introduced as an alternative type of discriminator D, differs from SeqGAN, it consists of two components, a feature extractor $feature(s; \phi_f)$ with CNN and a binary classification layer [3].

$$D_\phi(s) = \sigma(\phi_l^T F_{feature}(s; \phi_f)) = \sigma(\phi_l^T f) \tag{2.27}$$

where σ is a sigmoid function $\sigma(x) = \frac{1}{1+e^z}$, the feature extractor learn and form a high dimension feature representation $f = F_{feature}(s; \phi_f)$ where s is denoted as sequence, such information leaks to the generator G_θ as a guiding signal to alleviate the sparsity signal during the training process [3].
A hierarchical architecture has been introduced in generator part. Including two generators: A Manager and a Worker. The Manager is an LSTM model which takes responsibility for taking with leaked feature vector f_t at each time step t and outputs a goal vector g_t [3]. It can be expressed as following equations:

$$\hat{g}_t, H_t^M = M(f_t, h_{t-1}^M; \theta_m) \tag{2.28}$$

$$g_t = \hat{g}_t / \|\hat{g}_t\| \tag{2.29}$$

where $M(\cdot; \theta_m)$ is the Manager with parameter θ_m, in order to obtain current time goal vector $\hat{g}_t$ and hidden state h_{h-1}^m, the Manager needs to feed with current time step feature vector f_t and previous time hidden state h_{t-1}^m, to stabilize the model behavior, goal vector needs to be normalized as Equation (2.29). The feature subgoal vector g_t needs to go through a linear projection $\psi(g_t)$ model with weight matrix W_ψ to produce a guiding signal w_t in order to govern the Worker's generated sequence[3]. It can be expressed as following Equation (2.30).

$$w_t = \psi(\sum_{i=1}^{c} g_{t_i}) = \mathrm{W}_\psi(\sum_{i=1}^{c} g_{t-i}) \tag{2.30}$$

the Worker is similar to the Manger denoted as $W(.; \theta_w)$ with parameter θ_w, fed with current input x_t and previous hidden state h_{t-1}^w, outputs O_t which contains all current generated token and current hidden state h_t^w, the outputs O_t then further multiply with matrix w_t which is a guiding signal from Manager[3].

$$O_t, h_t^w = \mathcal{W}(x_t, h_{t-1}^W; \theta_\omega) \tag{2.31}$$

$$G_\theta(\cdot | s_t) = \text{softmax}(O_t \cdot \omega_t / \alpha) \tag{2.32}$$

In the training process, the Worker is updated by applying a policy gradient algorithm followed by SeqGan (Yu et al)[3] and loss is defined as:

$$L_w^{pre} = \mathbb{E}_{s_{t-1} \sim G}[\sum_{x_t} W(x_t | s_{t_1}; \theta_w)] \tag{2.33}$$

$$L_w^{adv} = \mathbb{E}_{s_{t-1} \sim G}[\sum_{x_t} r_t^I W(x_t | s_{t_1}; \theta_w)] \tag{2.34}$$

where r_t^I is the intrinsic reward can be described as:

$$r_t^I = \frac{1}{c} \sum_{i=1}^{c} d_{cos}(f_t - f_{t-i} - g_{t-i}) \tag{2.35}$$

Since these mathematical variables are pretty similar as other equations and hence, it will be explained later in eq 2.36.

In the pre-train and adversarial training stage. the Manager's parameters are updated though policy gradient as well which can be expressed as following accordingly[3]:

$$\nabla_{\theta_m}^{pre} g_t = \nabla_{\theta_m} d_{cos}(f_{t+c} - f_t, g_t(\theta_m)) \tag{2.36}$$

$$\nabla_{\theta_m}^{adv} g_t = -Q_f(s_t, g_t) \nabla_{\theta_m} d_{cos}(f_{t+c} - f_t, g_t(\theta_m)) \tag{2.37}$$

where $f_{t+c} - f_t$ calculates the feature representation f_t difference after c time steps, $g_t(\theta_m)$ is the goal vector, the output from Manager which has been described in Equation (2.29) and (2.28) $Q_f(s_t, g_t) = \mathbb{E}[r_t]$ is the expected reward can be calculate by applying Monte Carlo approach which has been mentioned in Section 2.10. d_{cos} calculates the cosine similarity of two sequences which can be represented as two vector $\overrightarrow{a}$ and $\overrightarrow{b}$ respectively, equation can be described as [13]:

$$cos\theta = \frac{\overrightarrow{a} \cdot \overrightarrow{b}}{\|\overrightarrow{a}\| \left\|\overrightarrow{b}\right\|} \tag{2.38}$$

In both pre-training and adversarial training stage, the loss of discriminator is a cross entropy which can be expressed as:

$$L_D = -(y\text{log}(p) + (1-y)\text{log}(1-p)) \tag{2.39}$$

Where p is the confidence score which is the probability value from discriminator. y is the true label which included synthetic data is labeled as 0, and real data is labeled as 1.

2.2.5.3 Other Issues

N-gram

N-gram is a probabilistic language model that it used to estimate the probability of next occurring word in a sequence of words. If the input is a sentence which sequence of each word, N-gram model can calculate the probability of the sentence, in other words, the joint probability of these words in the sentence. Commonly, in text classification problem usually use Bi-gram(N=2) and Tri-gram(N=3), for instance:

Bi-gram: (Machine learning), (Deep learning), (computer science)
Tri-gram: (Deep machine learning),(Artificial neural network)

Suppose there is a sentence denoted as f comprises of n word $f = (v_1, v_2, ...v_n)$, each word v_i depends on the effect of the occurrence probability from the first word v_1 to its previous word v, then the totally probability of this sentence is:

$$p(f) = p(v_2|v_1)p(v_3|v_1v_2)...p(v_n|v_{n-1}...v_2v_1) \tag{2.40}$$

Embedding Layer Approach

Embedding approach widely used in NLP field, it can convert discrete variables into a continuous vector representation. The most significant advantage of such a method is it can represent complex contexts. Embedding method can capture spatial information from high dimension and compress it into lower densely dimension. Compare to other conventional methods such as n-gram which mentioned in Section 2.4 and one-hot encoding, if the number of variable N increase, the total dimension of matrix representing would grow exponentially, this encounter dimensionality problem and leads to the curse of dimensionality, with embedding approach, the number of parameters grows only linearly and hence, it can efficiently prevent such issue [14].

Reinforcement Learning and Policy Gradient

Reinforcement learning is a particular type of algorithms differs from supervised learning refers to how good an agent interacts in an environment in order to maximize cumulative reward r [15]. The agent can have various state $(s \in S)$ with different action $(a \in A)$, in order to optimize the action to get the maximum reward, a policy $\pi(s)$ defines to describe agent's behavior as a strategy in a given state. During the action-optimization training, the agent performs a number of actions to explore the environment followed by different policy $\pi(s)$, the agent should not only consider the reward of the generated sequence but also consider the long-term benefits. In order to make decision for the best current action, the agent accumulates all the expected future reward includes the current time-step reward, the larger accumulative reward means the better action it is, even the current state reward is negative and hence, reinforcement learning has capable of sacrificing immediate reward to learn long-term strategy.

Policy gradient approach is a widely used technique that aims to update optimal policy $\pi(s)$ with long-term feedback through gradient descent. It does not update the model's parameter through back-propagation directly. Instead of it uses reward to update the possibility of the selected action. The probability of the corresponding action will be increased with positive long-term reward, a negative reward will decrease the likelihood of action which will be chosen in the next time. In mathematical terms, it can be described as Equation (2.41) [16]:

$$J(\theta) = \sum_{s\in S} d^{\pi}(s)V^{\pi}(s) = \sum_{s\in S} d^{\pi}(s) \sum_{a\in A} \pi(a|s)Q^{\pi}(s,a) \tag{2.41}$$

where, $d^{\pi}(s)$ represents the probability in state s following by policy π, the expected reward $J(\theta)$ equals to the averaging of all the value of state followed by a policy π, the state value function $V(s)$ can be rewritten as averaging all the action-value followed with policy π.

2.3 Methods for Learning Hyper-Parameters

2.3.1 Vanishing Gradient Descent

The vanishing gradient problem limits the development of neural networks such as RNN. To briefly explain the phenomenon, we introduce a single feed-forward neuron network with multiple layers. The figure 2.11 illustrates the structure of a multiple layer network.

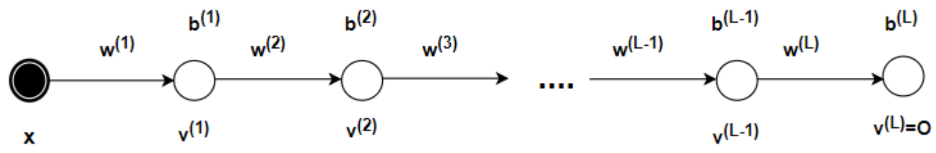


Figure 2.11: Illustration depicting the principle of back-propagation

In layer i,each layer's weights and biases denoted as $L^{(i)}$ $w^{(i)}$ and $b^{(i)}$ accordingly, the output and 'ground true' target are denoted as $o^{(i)}$ and $t^{(i)}$ respectively.
Using the feed-forward principle which was presented in Section 2.2.1, the output O will be:

$$O = V^{(i)} = f(w^{(i)}(f(w^{(i-1)}...f(w^{(2)}f(w^{(1)}x + b^{(1)}) + b^{(2)})..b^{(i-1)}) + b^{(i)}) \tag{2.42}$$

where f is the activation function,to simplify the notation the choice of activation function is not specified. The impact of each layer is calculated using the chain rule by:

$$\frac{\partial V^{(i)}}{\partial V^{(i-1)}} = f'(c^{(i)})w^{(i)} \tag{2.43}$$

$$\frac{\partial V^{(i)}}{\partial V^{(i-2)}} = \frac{\partial V^{(i)}}{\partial V^{(i-1)}}\frac{\partial V^{(i-1)}}{\partial V^{(i-2)}} = f'(c^{(i)})w^{(i)}f'(c^{(i-1)})w^{(i-1)} \tag{2.44}$$

$$\vdots$$

where $c^{(i)} = w^{(i)}V^{(i-1)} - b^{(i)}$. This yields the error expression updating for each layer:

$$E^{(i)} = [t - V^{(i)}(x)]f'(c^{(i)}) \prod_{j=i}^{i+1} [w^{(i)}f'(c^{(j-1)})] \tag{2.45}$$

Intuitively, if the terms $w^{(i)} f'(c^{(j-1)})$ are either less or larger than one, then the error values either vanishes or explodes rapidly when i decreases. Thus the error updating for each layer will be very large or small. This leads directly to the instability of the network and causes the vanishing gradient descent problem [17].

2.3.2 Dropout

Deep neural network's non-linear properties make it adaptable to fit complex relationships between inputs and outputs. With this characteristic, model will learn the spurious noise which only exists in training dataset but not in test data even though both have the same distribution, and leads to an over-fitting problem [18].

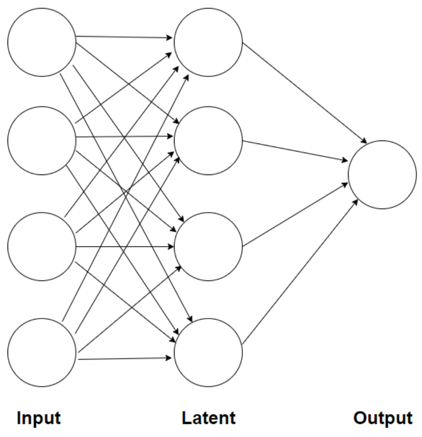


Figure 2.12: Vanilla full connected network.

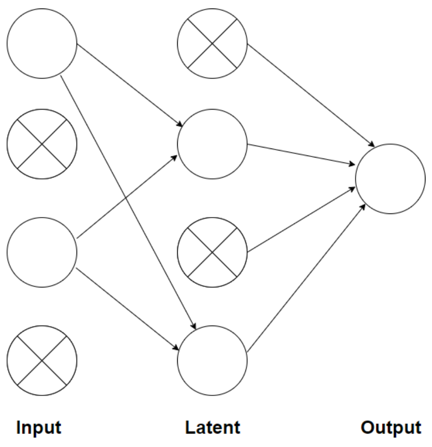
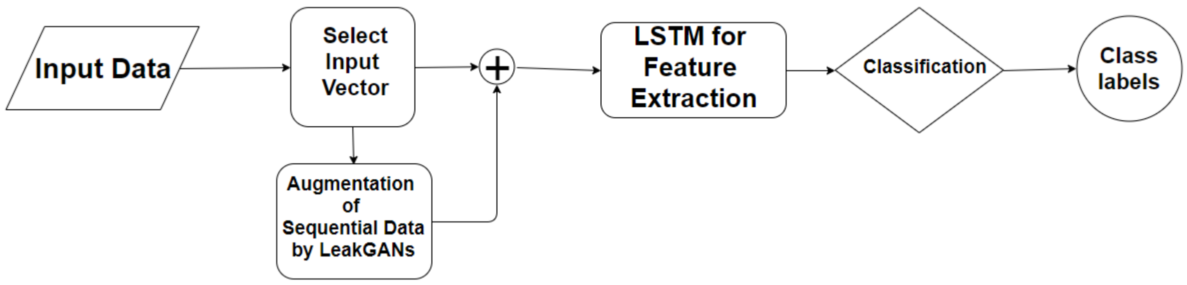


Figure 2.13: Full connected with fifty percent dropout.

Dropout is an efficient way to address this issue. The underlying idea is to randomly remove some neurons from the network temporarily with a certain percentage, this decrease the complexity of model, make the model more robust against noise.

3
Towards Automated Infotainment Systems: a LSTM-based Approach.

The aim of this project is to improve the efficiency for automated testing via implementing machine learning approach for automating an infotainment system. This chapter describes the method which has been studied and developed in this project. The data pre-processing is needed. The rationale for enhancing the classification performance using GAN model is presented as well.



Input Data
Select Input Vector
Augmentation of Sequential Data by LeakGANs
LSTM for Feature Extraction
Classification
Class labels

Figure 3.1: Block diagram illustrating the use of LSTM and LeakGAN model in the detection system

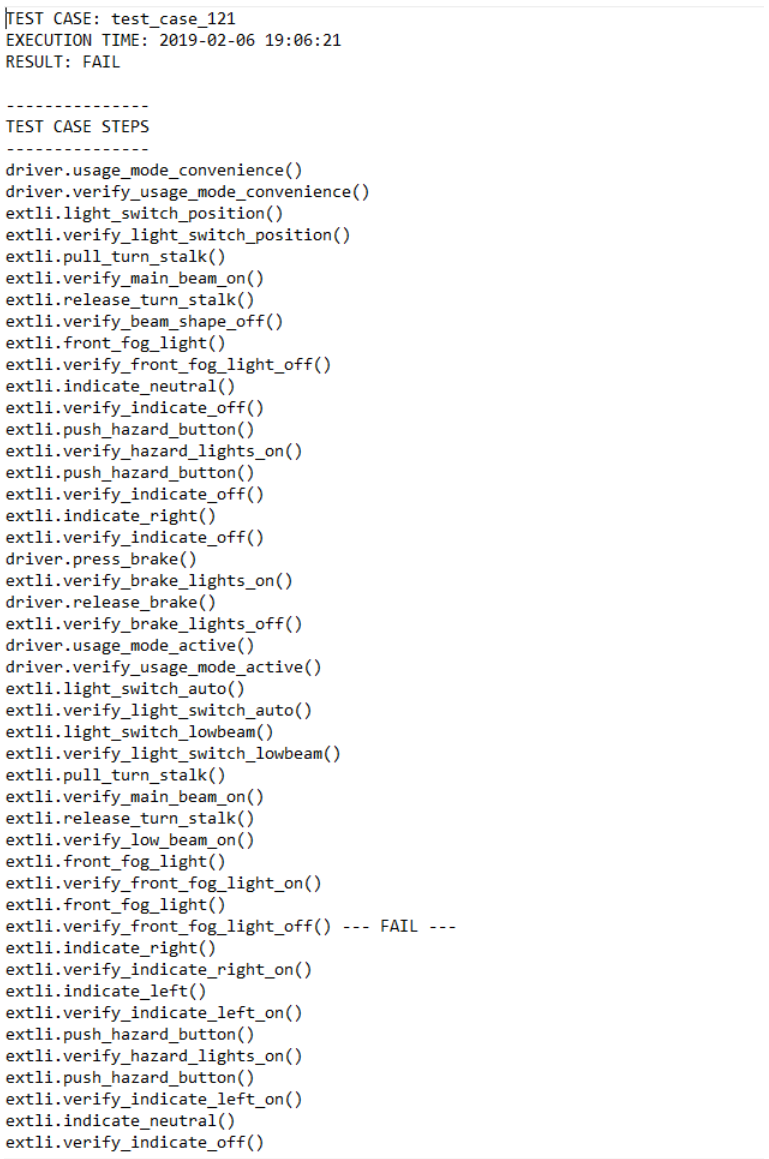
3.1 LSTM for the Sequential Prediction

The system has been implemented from scratch with Keras using Tensorflow as backend. Figure 3.1 shows the flowchart of LSTM detection system. The coming subsection is presented and followed by this block diagram.

3.1.1 Input Data

Log files contain detailed information about each executed automated test case. Up to at least 15000 log-files has been produced and collected by CEVT and has been provided to this thesis project. The log-files provided contains the input and output of the automated test execution as data concerning executed steps, timestamps, pass/fail outcome and error key for the failed case of each action.

The log-files consists of two parts, the first part is the TEST CASE STEPS parts. This part contains which test function will be executed in a certain sequence broadly without much detail information. In order to initialize the test case, a certain driving mode needs to be chosen first. Figure 3.2 shows what it looks like:

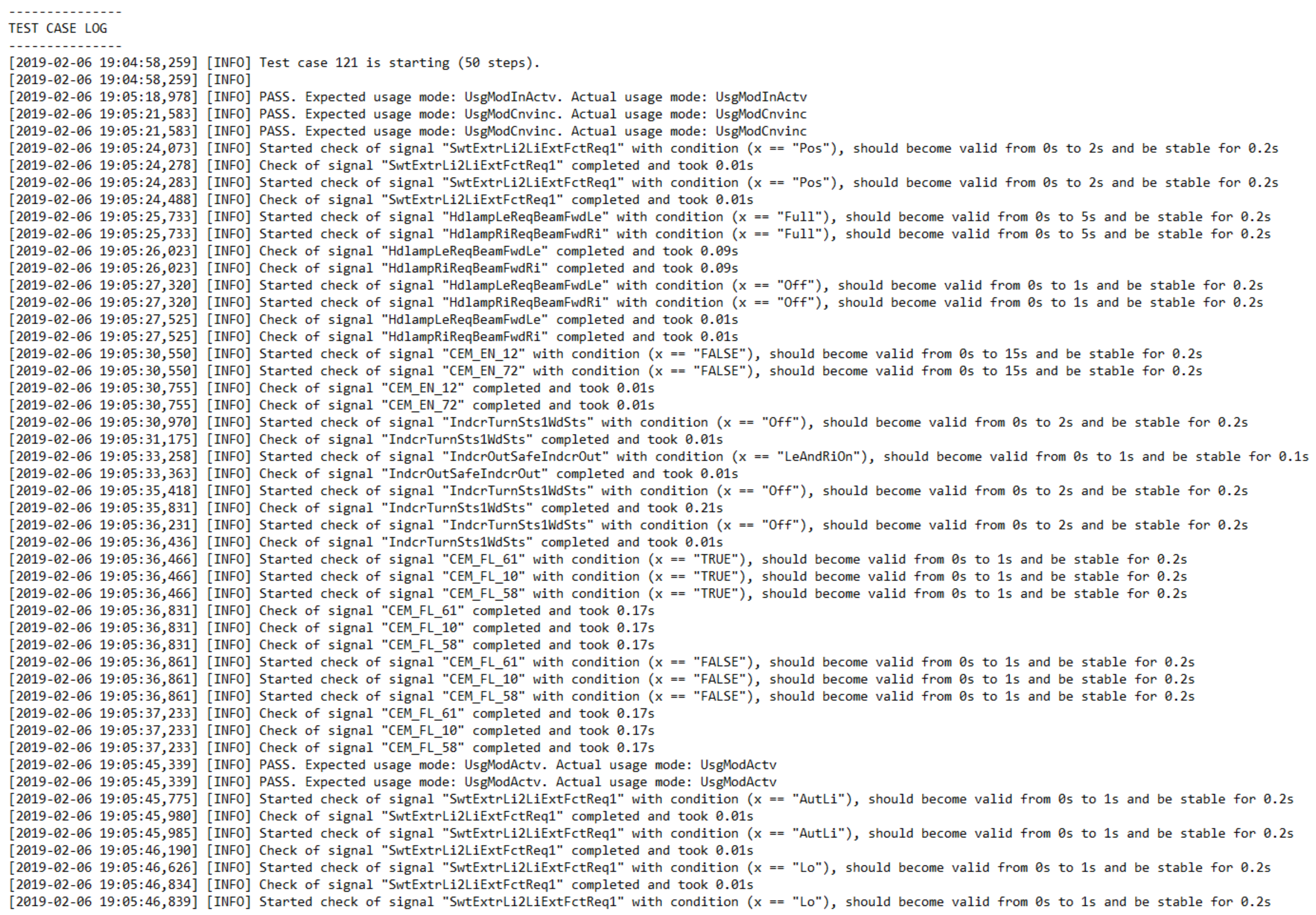


```
TEST CASE: test_case_121
EXECUTION TIME: 2019-02-06 19:06:21
RESULT: FAIL

--------------
TEST CASE STEPS
--------------
driver.usage_mode_convenience()
driver.verify_usage_mode_convenience()
extli.light_switch_position()
extli.verify_light_switch_position()
extli.pull_turn_stalk()
extli.verify_main_beam_on()
extli.release_turn_stalk()
extli.verify_beam_shape_off()
extli.front_fog_light()
extli.verify_front_fog_light_off()
extli.indicate_neutral()
extli.verify_indicate_off()
extli.push_hazard_button()
extli.verify_hazard_lights_on()
extli.push_hazard_button()
extli.verify_indicate_off()
extli.indicate_right()
extli.verify_indicate_off()
driver.press_brake()
extli.verify_brake_lights_on()
driver.release_brake()
extli.verify_brake_lights_off()
driver.usage_mode_active()
driver.verify_usage_mode_active()
extli.light_switch_auto()
extli.verify_light_switch_auto()
extli.light_switch_lowbeam()
extli.verify_light_switch_lowbeam()
extli.pull_turn_stalk()
extli.verify_main_beam_on()
extli.release_turn_stalk()
extli.verify_low_beam_on()
extli.front_fog_light()
extli.verify_front_fog_light_on()
extli.front_fog_light()
extli.verify_front_fog_light_off() --- FAIL ---
extli.indicate_right()
extli.verify_indicate_right_on()
extli.indicate_left()
extli.verify_indicate_left_on()
extli.push_hazard_button()
extli.verify_hazard_lights_on()
extli.push_hazard_button()
extli.verify_indicate_left_on()
extli.indicate_neutral()
extli.verify_indicate_off()
```

Figure 3.2: Figure illustrates the log-file's Test Case Steps part

The second part is the TEST CASE LOG. It detailed documents i.e. test step length and timestamps for executed test function with specific condition and expected acceptance signal. If an error occurs during the testing process, An error message which includes keyword will be recorded in order to facilitate troubleshooting later. Figure 3.3 illustrates what TEST CASE LOG part looks like:



```
---------------
TEST CASE LOG
---------------
[2019-02-06 19:04:58,259] [INFO] Test case 121 is starting (50 steps).
[2019-02-06 19:04:58,259] [INFO]
[2019-02-06 19:05:18,978] [INFO] PASS. Expected usage mode: UsgModInActv. Actual usage mode: UsgModInActv
[2019-02-06 19:05:21,583] [INFO] PASS. Expected usage mode: UsgModCnvinc. Actual usage mode: UsgModCnvinc
[2019-02-06 19:05:21,583] [INFO] PASS. Expected usage mode: UsgModCnvinc. Actual usage mode: UsgModCnvinc
[2019-02-06 19:05:24,073] [INFO] Started check of signal "SwtExtrLi2LiExtFctReq1" with condition (x == "Pos"), should become valid from 0s to 2s and be stable for 0.2s
[2019-02-06 19:05:24,278] [INFO] Check of signal "SwtExtrLi2LiExtFctReq1" completed and took 0.01s
[2019-02-06 19:05:24,283] [INFO] Started check of signal "SwtExtrLi2LiExtFctReq1" with condition (x == "Pos"), should become valid from 0s to 2s and be stable for 0.2s
[2019-02-06 19:05:24,488] [INFO] Check of signal "SwtExtrLi2LiExtFctReq1" completed and took 0.01s
[2019-02-06 19:05:25,733] [INFO] Started check of signal "HdlampLeReqBeamFwdLe" with condition (x == "Full"), should become valid from 0s to 5s and be stable for 0.2s
[2019-02-06 19:05:25,733] [INFO] Started check of signal "HdlampRiReqBeamFwdRi" with condition (x == "Full"), should become valid from 0s to 5s and be stable for 0.2s
[2019-02-06 19:05:26,023] [INFO] Check of signal "HdlampLeReqBeamFwdLe" completed and took 0.09s
[2019-02-06 19:05:26,023] [INFO] Check of signal "HdlampRiReqBeamFwdRi" completed and took 0.09s
[2019-02-06 19:05:27,320] [INFO] Started check of signal "HdlampLeReqBeamFwdLe" with condition (x == "Off"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:27,320] [INFO] Started check of signal "HdlampRiReqBeamFwdRi" with condition (x == "Off"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:27,525] [INFO] Check of signal "HdlampLeReqBeamFwdLe" completed and took 0.01s
[2019-02-06 19:05:27,525] [INFO] Check of signal "HdlampRiReqBeamFwdRi" completed and took 0.01s
[2019-02-06 19:05:30,550] [INFO] Started check of signal "CEM_EN_12" with condition (x == "FALSE"), should become valid from 0s to 15s and be stable for 0.2s
[2019-02-06 19:05:30,550] [INFO] Started check of signal "CEM_EN_72" with condition (x == "FALSE"), should become valid from 0s to 15s and be stable for 0.2s
[2019-02-06 19:05:30,755] [INFO] Check of signal "CEM_EN_12" completed and took 0.01s
[2019-02-06 19:05:30,755] [INFO] Check of signal "CEM_EN_72" completed and took 0.01s
[2019-02-06 19:05:30,970] [INFO] Started check of signal "IndcrTurnSts1WdSts" with condition (x == "Off"), should become valid from 0s to 2s and be stable for 0.2s
[2019-02-06 19:05:31,175] [INFO] Check of signal "IndcrTurnSts1WdSts" completed and took 0.01s
[2019-02-06 19:05:33,258] [INFO] Started check of signal "IndcrOutSafeIndcrOut" with condition (x == "LeAndRiOn"), should become valid from 0s to 1s and be stable for 0.1s
[2019-02-06 19:05:33,363] [INFO] Check of signal "IndcrOutSafeIndcrOut" completed and took 0.01s
[2019-02-06 19:05:35,418] [INFO] Started check of signal "IndcrTurnSts1WdSts" with condition (x == "Off"), should become valid from 0s to 2s and be stable for 0.2s
[2019-02-06 19:05:35,831] [INFO] Check of signal "IndcrTurnSts1WdSts" completed and took 0.21s
[2019-02-06 19:05:36,231] [INFO] Started check of signal "IndcrTurnSts1WdSts" with condition (x == "Off"), should become valid from 0s to 2s and be stable for 0.2s
[2019-02-06 19:05:36,436] [INFO] Check of signal "IndcrTurnSts1WdSts" completed and took 0.01s
[2019-02-06 19:05:36,466] [INFO] Started check of signal "CEM_FL_61" with condition (x == "TRUE"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:36,466] [INFO] Started check of signal "CEM_FL_10" with condition (x == "TRUE"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:36,466] [INFO] Started check of signal "CEM_FL_58" with condition (x == "TRUE"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:36,831] [INFO] Check of signal "CEM_FL_61" completed and took 0.17s
[2019-02-06 19:05:36,831] [INFO] Check of signal "CEM_FL_10" completed and took 0.17s
[2019-02-06 19:05:36,831] [INFO] Check of signal "CEM_FL_58" completed and took 0.17s
[2019-02-06 19:05:36,861] [INFO] Started check of signal "CEM_FL_61" with condition (x == "FALSE"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:36,861] [INFO] Started check of signal "CEM_FL_10" with condition (x == "FALSE"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:36,861] [INFO] Started check of signal "CEM_FL_58" with condition (x == "FALSE"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:37,233] [INFO] Check of signal "CEM_FL_61" completed and took 0.17s
[2019-02-06 19:05:37,233] [INFO] Check of signal "CEM_FL_10" completed and took 0.17s
[2019-02-06 19:05:37,233] [INFO] Check of signal "CEM_FL_58" completed and took 0.17s
[2019-02-06 19:05:45,339] [INFO] PASS. Expected usage mode: UsgModActv. Actual usage mode: UsgModActv
[2019-02-06 19:05:45,339] [INFO] PASS. Expected usage mode: UsgModActv. Actual usage mode: UsgModActv
[2019-02-06 19:05:45,775] [INFO] Started check of signal "SwtExtrLi2LiExtFctReq1" with condition (x == "AutLi"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:45,980] [INFO] Check of signal "SwtExtrLi2LiExtFctReq1" completed and took 0.01s
[2019-02-06 19:05:45,985] [INFO] Started check of signal "SwtExtrLi2LiExtFctReq1" with condition (x == "AutLi"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:46,190] [INFO] Check of signal "SwtExtrLi2LiExtFctReq1" completed and took 0.01s
[2019-02-06 19:05:46,626] [INFO] Started check of signal "SwtExtrLi2LiExtFctReq1" with condition (x == "Lo"), should become valid from 0s to 1s and be stable for 0.2s
[2019-02-06 19:05:46,834] [INFO] Check of signal "SwtExtrLi2LiExtFctReq1" completed and took 0.01s
[2019-02-06 19:05:46,839] [INFO] Started check of signal "SwtExtrLi2LiExtFctReq1" with condition (x == "Lo"), should become valid from 0s to 1s and be stable for 0.2s
```

Figure 3.3: Figure illustrates the log-file's Test Case Log part

There are totally 39 different test functions used to generate the test cases. These functions are the exterior light part of the vehicle, for instance: brake light, rear fog light, right indicator, etc. These indicator components are connected together to a server, the server then sends the test signal which following the test sequence from generated test-case. Each component has corresponding sensor will send back the signal to the server, the server will then verify whether the component works regularly.

All the test cases have been generated and tested in the Hardware-In-the-Loop(HIL) system and been labeled as Passed and Failed. HIL simulation methodology is an useful and effective manner in automating testing. It provides great compatibility for components replacing and reduction of time for the development costs. Compare to conventional pure numerical simulation without any hardware. HIL simulation system can produce significant better result and show the feasibility of practical implementation[19].

3.1.2 Selection of Input Vector

In order to extract the sequential information of the log-file, a rudimentary technique can be adapted is regular expression which has been studied and used to extract test sequence. Regular expression is a special sequence of characters that define a search pattern used to match string pattern and extract information from text such as log-file,text etc.

A general python script has been written in order to batch handle and extract test sequence from log-files and save as a CSV-file separately. In order to quantify the data, each test function has been assigned a unique value randomly from 1 to 39. Each case is converted to a string with various integer digits.

$$[39 \quad 1 \quad ...... \quad 22 \quad 14 \quad 8 \quad 3]$$

In order to feed input vector into a neural network, the first layer embedding required a fixed length input size , this means Zero Padding technique is needed to be applied, the Padding length is aligned with the test cases' maximum length which is 300.

$$\underbrace{[0 \quad 0 \quad ..... \quad 39 \quad 1 \quad ...... \quad 22 \quad 14 \quad 8 \quad 3]}_{\text{Length } = 300}$$

3.2 Sequence Data Learning and Prediction by LSTMs

In recent years, Python became one of the most popular languages for machine learning with its readability and versatility, as a highly extensible programming language. Python supports over a hundred thousand different libraries and has been well optimized and documented. In machine learning field, several top deep learning frameworks such as Pytorch, Tensorflow has been well supported. Therefore, in this project , it has been selected as primary language.

3.2.1 Architecture of LSTM

Figure 3.4 shows the structure of LSTM. It initializes with an input layer, and the input layer has shape (None, 300) which means it can receive any amount of inputs with variable-length sequence of 300-dimensional vectors since embedding layer need a fixed size of input which has been mentioned in Section 3.1.2. It sends one token each time step into the embedding layer, the embedding layer then maps of a discrete token to a vector of high dimension representation. Afterthat, the data goes though LSTM for learning required for subsequent .

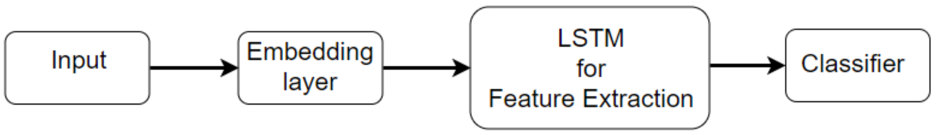


Figure 3.4: Lstm model starcture

3.2.2 Many to One

Figure 3.5 illustrates the many to one structure which has been used to perform the prediction. As can see, each LSTM cell can have a amount of circles, these circles represent the basic LSTM units which has been explained and illustrated in Section 2.8, In each LSTM cell, these units can be described as a feed-forward network and depend on the hyper-parameter setting. In each time step, the input sends data once at a time to the LSTM cell. In the final time step, the LSTM cell will send an output to the classifier to perform prediction. The classifier then returns a probability value between 0 and 1 which can be regarded as confident value, if the value is close to 1, it means the corresponding test cases has almost 100 percent confidence can be seen as passed test cases and vice versa. If there are two layers, then the LSTM cell sends hidden state not only to the next time step's LSTM cell of the current layer, but also sends it to the LSTM cell which is in next layer with current time step.

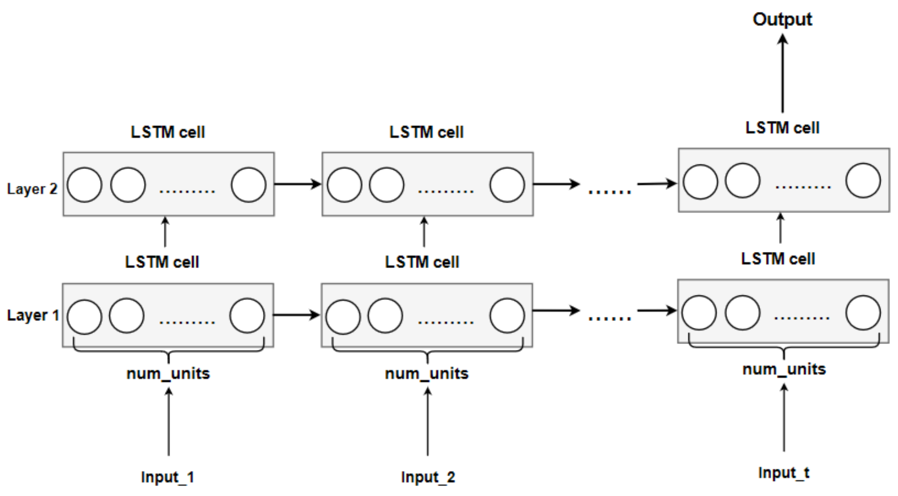


Figure 3.5: Lstm model detailed structure

3.3 Sequential Data Augmentation

Many methods have been tried and tested e.g. downsampling upsampling, class weight. In image processing, many tricks can be implemented to augment the data-set, e.g. mirror the image, do some rotation of the image, scale image etc. In NLP field. Synonym Replacement is most efficient one. for instance, if there is a sentence 'I love deep learning' . 'love' can be replaced with 'like'. The sentence still retains same segmentation, but the data distribution has been changed. None of the augmentation techniques which are mentioned above is suitable and applicable for such specific automating testing since each test function representation is totally irrelevant and independent.

One way might be feasible for data augmentation is to apply GAN to generate synthetic data through learning the data distribution from all failed test cases, since most information will be lost by downsampling the data-set and therefore, generate equally amount of failed cases data as passed test cases might be useful and solve this issue.
The code for data augmentation LeakGAN model was downloaded from '`https://github.com/CR-Gjx/LeakGAN/tree/master/Image%20COCO`' and has been tested with the experimented dataset to guarantee the program is executable and has same performance as it has been mentioned in the research paper. In this project, the code will be used as the main framework for further investigation.

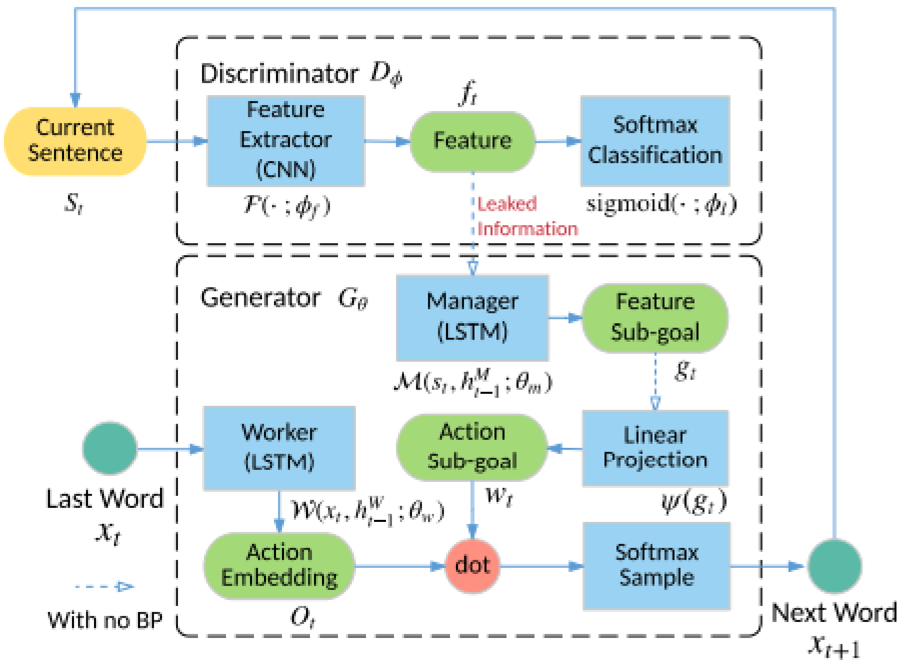


Figure 3.6: Figure illustrates the structure of LeakGAN[3]

.

3.3.1 Pseudo Algorithm for LeakGAN

One primary difference LeakGAN [3] introduced another generator Manager to alleviate the sparsity reward issue. Secondary, LeakGAN performs interleaved training in per-train process to mitigate model collapse issue, since it is a typical issue for GAN model.

The following table shows the algorithm of LeakGAN:

Algorithm 2: LeakGAN algorithm

```
Initialize W(Worker) M(Manager) D(Discriminator) with random weights;
Pre-train D using positive sample and negative sample with cross entropy;
Pre-train W and M using leaked information from D with Eq (2.33) and (2.36);
Conduct pre-training alternating until convergence ;
while Adversarial Training do
    for generator do
        Generate sequence $S_{1:T} = (s_1, ..., s_T)$
        for t in 1:T do
            Store Leaked information $f_t$ from D;
            Obtain $g_t$ from Manager with Eq (2.29) and (2.28);
            Get expected reward $Q(f_t, g_t)$ using Monte Carlo approach ;
            Update W and M parameter with Eq (2.34) and (2.37) respectively
    for discriminator do
        Generate negative samples using current W and M and concatenate
          positive samples;
        Train D for N iterations with Eq (2.27)
```

First, initialize the weights for generator including Worker W and Manager M and the Discriminator D from a Gaussian distribution $N(\mu = 0, \sigma = 1)$, with zero mean and one standard deviation. In the pretraining process, use cross-entropy as loss function to pretrain the Discriminator with positive samples, which is the real failed test cases and the negative samples which are produced from generator. Then pretrain the W and M using leaked information from D. These two pretraining steps perform alternatively until the model converges.

In adversarial training, the Generator first generates a complete sequence $S_{1:T}$ from starting time to final time step T as sample which uses to obtain the feedback from Discriminator later. In each time step t, it sends a partial sequence $S_{1:t} = (s_1, ..., s_t)$ to the discriminator in order to attain leaked information f_t. Furthermore, with Leak information, we can attain the feature sub-goal g_t from Manager via Eq (2.29) and (2.28). After that, it applies Monte Carlo search method to obtain expected Reward $Q(f_t, g_t)$ as Error and adopt Reinforcement Learning method Policy Gradient to perform back-propagation to update generator's parameter. Such adversarial training process repeats until LeakGAN converges.

3.4 Dataset

The data-set which provided by CEVT for test case prediction has severe imbalance problem, Table 3.2 describes the total number of original data in each class:

# Dataset	**Class "Passed"**	**Class "Failed"**
16894	15988	906
100%	94.64%	5.36%

Table 3.1: Table shows totally number of dataset which use to analyse in this project and ratio between class "Failed" and "Passed"

In order to deal with it, several manners has been tried, we found by adapting Down-sampling technique though downsample the majority data class which is the Passed test cases will help the model converge faster. Another word, randomly sample from passed test cases in order to balance the dataset as 50/50 which can achieve the best performance.

3.4.1 Training Dataset

There are totally 1810 data sequences in the training dataset after applying down-sampling, the dataset is randomly split with ratio 64%, 16% and 20% for training, validation, and testing respectively which shows in the table:

Training	Validation	Testing	Total
1158	290	362	1810
64%	16%	20%	100%

Table 3.2: Table shows the ratio of the dataset

3.4.2 Augmentated Data by LeakGAN

The dataset for training process is all the failed test cases. However, since the length of input sequence diverse among each date and the LeakGAN can only output a fixed length which is determined by hyper-parameter. If the length of output is set as maximum length 300, then the time of training process which applies roll-out policy described in Section 2.2.5.1 grows exponentially with long sequence and these type of dataset only acounts for 0.5% of total dataset which has been shown in Figure 3.7 and therefore, enable to save the training time and also stabilize the training process, with empirical test we found set input length equal to 150 is an appropriate choice.

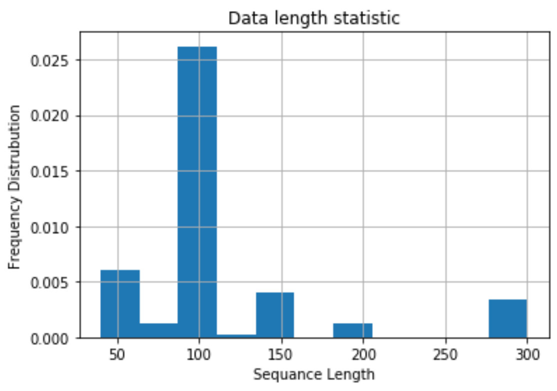


Figure 3.7: Figure illustrate the frequency of sequence length

Therefore, a small fraction of data which the length of a sequence is large than 150 is discarded.

4
Experimental Results

This chapter is mainly divided into two parts. The first section shows the performance of LSTM with down-sampling technique. Several study cases have been designed and experimented to achieve the best model prediction performance. The second section describes the results of applying LeakGAN for data augmentation, the experiment outcome is outlined and presented.

4.1 Evaluation Metric

This section describes three different metrics which use to assess the model performance.

4.1.1 LSTM Model's Accuracy Metric

The output from classifier is a confident value between 0 and 1, by setting a threshold which is equivalent to 0.5, if the output is higher than 0.5, then the data is labeled as passed cases and vice versa, the expression can be described as following, where O is the output from classifier:

$$\text{Label} = \begin{cases} \text{Pass} & \text{if} \quad O > 0.5 \\ \text{Fail} & \text{Otherwise} \end{cases} \tag{4.1}$$

4.1.2 Confusion Matrix

Confusion matrix (CM) is a common way used to visualizes the performance measurement for individual classes. Each row represents the predicted class, and each column represents the true class. For a binary classification then it will be a two by two matrix with four different element: True Positive(TP), True Negative(TN), False Positive(FP), False Negative(FN). Figure 4.1 shows the visualization of CM.

4.1.3 Receiver Operating Characteristic

Receiver operating characteristic approach (ROC) is another useful tool to evaluate prediction quality for 2 classes. It shows the curve in continuous measurement and describes the ratio between FP and TP across a range of various classification thresholds. The Y-axis represents the true positive rate (TPR) and x-asix represents the false positive rate (FPR), the ideal target is to set $TPR = 1$ and $FPR = 0$, which is the (0,1) point in the figure, this means all the classification prediction

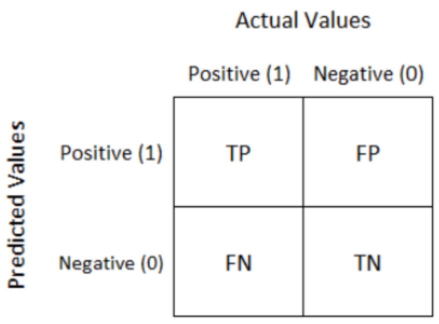


Figure 4.1: Confusion matrix

is correct. Since the company wants to filter out passed test as many as possible to save testing time meanwhile still retain high accuracy for TN prediction. Thus it is informative to compare model performance and set the best optimal decision threshold.

Area under curve (AUC) is the area under the ROC curve, another measurement metric to evaluate the classifier performance. It is a probability value between 0 and 1. Typically, The larger of AUC values indicate the better of model's performance it is.

4.1.4 2-D Histogram

To verify whether the output from LeakGAN converges to real data distribution, both research paper [12, 3] have introduced en Oracle model G_{oracle} to produce the real data distribution. The benefit of having such oracle is that firstly, the model is initialized with weight from Gaussian distribution and it can produce data-set which can be represented as 'true' data distribution. Secondly, it is easy to evaluate the performance of the generative model G_θ. Since the Oracle model represents the real data distribution, in each time via picking some samples generator G_θ and calculate the loss by adopting negative log-likelihood(NLL)[12]. However, in reality, there is never such model exists and hence, it is hard to verify whether the generated sequence from generator G_θ converges to true data distribution.

One alternative way to indirectly observe data property is to check the transfer probability density via plotting a 2D-histogram. In mathematics, Suppose the probability P from current time steps j to the next time step K is denoted as $Pr(k|j) = P_{j,K}$, then it can be described as a square matrix.

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \dots & P_{1,k-1} & P_{1,k} \\ P_{2,1} & P_{2,2} & \dots & P_{2,k-1} & P_{1,k} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ P_{j,1} & P_{j,2} & \dots & P_{j,k-1} & P_{j,k} \end{bmatrix} \tag{4.2}$$

where summation of totally transition probability from time steps j to all the next time step must be equal to 1:

$$\sum_{j=1}^{S} P_{i,j} = 1 \tag{4.3}$$

4.2 Setup

Both LeakGAN and LSTM-model were trained on a desktop computer with Windows 10 system and Anaconda platform. Table 4.2 shows the hardware and software specification which have been used in this project.

Graphic card	Palit GeForce GTX 1070 Ti Dual 8GB
Processor	Intel Core i5 2500K 3,3GHz
Memory	Corsair 16GB DDR3 CL10 1600MHz
Operating System	Microsoft Windows 10 Home
CUDA	10.0.130
CuDNN	7.3.1
Tensorflow	1.13.1
Pytorch	1.0.1
Keras	2.2.4

Table 4.1: Table shows the computer specification which used for training

4.3 Test Performance on LSTMs

4.3.1 Hyper-Parameter Fine-Tuning

We extensively tune the hyper-parameter with different cases empirically. This section describes the best results of hyper-parameter optimization

Embedding dimension. As mentioned in Section 2.2.5.3, Embedding approach represents each input variable as a distributed value in an n-dimensional space. Several various input value has been tested, we found that value between 40 and 120 can obtain the best results. With higher input dimension would make the model unstable and harder to perform gradient descent. the value of loss function would fluctuate vastly. Any significant lower and higher value will obtain worse accuracy performance. This matches the number of test function which means model prefer sparse vector instead of low dense dimension vector.

Learning rate. With the various experiment, a relative bigger initiation learning rate would help model fast converge to a sub-optimal point. Instead of using a fixed learning rate, vary the learning rate over the training process helps model further converge and significantly improve the accuracy. We found around 0.03 is an appropriate value from the start, with patience 4 and decrease factor 0.7.

One mishap is worth to be mentioned is that when setting learning rate around 4×10^{-4}, the model can obtain even higher performance on training accuracy, but model completely failed to perform any prediction, this is due to a too small learning rate may lead the model stuck with sub-optimal point and loss the generalization capability.

Early stopping approach is employed as a call back function, through empirical test the loss function is monitored with patient 7 can obtain the best results and prevent over-fitting problem.

Dropout: Different value of parameter dropout has been tested. We found 0.2 is the best value. With value below 0.15, the model would have severe over-fitting. With higher value, the model will have lower accuracy performance.

4.3.2 Training Results

In order to achieve the highest prediction performance for prediction system, this needs to be done mainly by empirical test, and hence, a series of tests are conducted and table 4.2 shows the several representative cases with various layer units and structure which have been designed and studied. In the column of LSTM indicates the number of units for each layer. A checkpoint strategy has been used to record the best model during the training process, the train and test accuracy in the following table represents the highest accuracy among the training epochs. All these cases has been conducted at least three times and pick the best results in order to eliminate the assessment bias.

Case Study	Layer	LSTM units	Train Accuracy	Test Accuracy
1	1	64	0.832	0.817
2	1	128	0.833	0.787
3	1	256	0.866	**0.855**
4	2	64/32	**0.868**	0.850
5	2	128/64	0.540	0.516

Table 4.2: Table illustrates the results of different study case

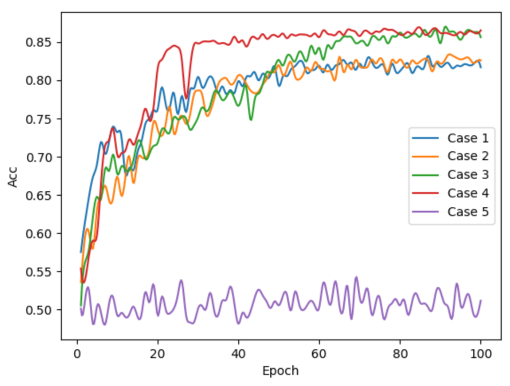


Figure 4.2: Training Accuracy

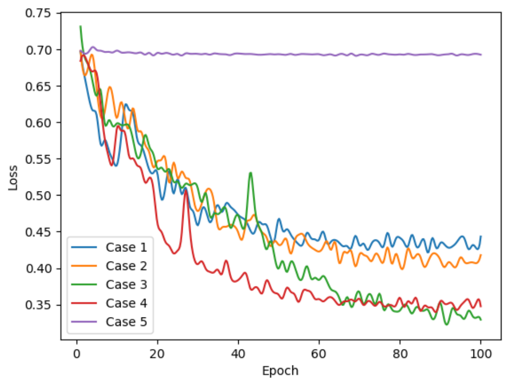


Figure 4.3: Training Loss

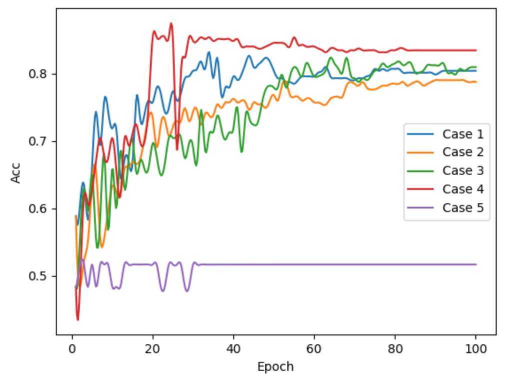


Figure 4.4: Validation Accuracy

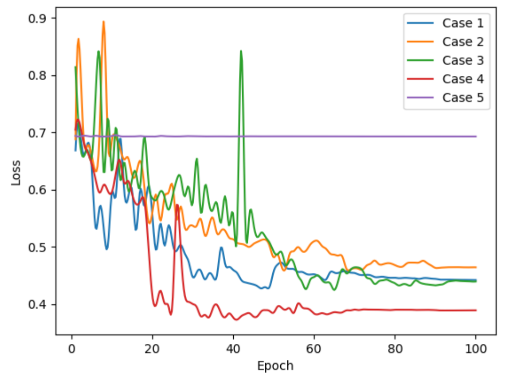


Figure 4.5: Validation Loss

Case study four shows that with two LSTM layers which have 64 and 32 units accordingly, the model can achieve the highest performance on training accuracy. However case study three can achieve slightly lower on training accuracy but relatively higher accuracy on the test data-set. Notes that in order to eliminate all the uncertainty factor which has been discussed in the Section 4.3.3, a random seed is applied to split the data deterministic randomly and hence, all the test data-set is precisely the same. Since the training dataset is pretty small with only around 1800, thus it is intuitively that with a less complicated model, it can achieve better performance to avoid the over-fitting problem. There are some other study cases has been tried. For instance, a much more sophisticated model or a much less complicated model, but the value of loss is either fluctuate vastly or hardly to conduct gradient descent. One interesting phenomenon that needs to be mentioned is that Case study five, the model seems totally failed to fit the data-set.

Figure 4.6 illustrates classification performance in confusion matrix. As can see, the classifier seems can distinguish more actual classes “Pass” and “Fail” cases.

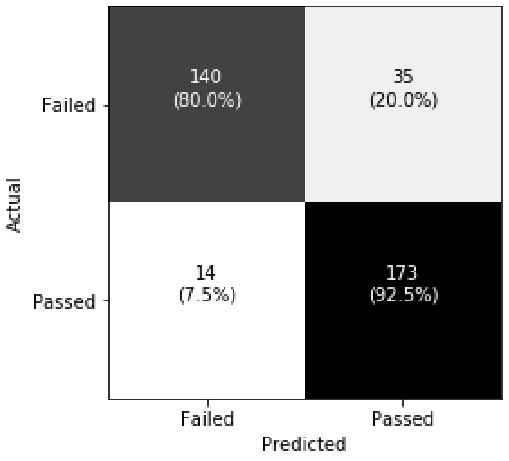


Figure 4.6: The confusion matrix for case study 3

There are totally 362 samples used to evaluate the classification performance. Each row represents the true label, there are 175 cases in what the actual label was Failed, and 187 cases which the actual label was passed. Each column represents the predicted label. The color in each cell shows as a gray scale represents the ratio of each class from white zero percent to the black which is hundred percent

4.3.3 Discussions

In order to maximum minimize the interference of various factors for the training process, a pseudo-random number generator is used to initialize model weight and split data-set randomly. In stead of performing experiment repeatedly, a pseudo random number generator makes ‘random’ sequence become deterministic. This means whether split a train test data-set, it will get exactly same distribution, this can further eliminate any uncertainty and avoid variation in training process, ensure performance on each experiment is as close as possible.

4.3.4 Threshold Optimization

In order to optimizing the LSTM detection system’s performance since our priority is to filter out all potential “ Pass” class to save testing time, while still retain high classification accuracy on the actual “Fail” class and hence. ROC cruve which has been mentioned in Section 4.1.3 is used for the analysis.
where Y-axis represents the true positive rate(TPR) equal to $\frac{TP}{P}$, X-axis represents the false positive rate(FPR) which is equivalent to $\frac{FP}{N}$, the curve is constructed by setting different thresholds and show the relation of various performance, the diagonal dot line represents the threshold of accuracy performance which is 50/50, in other words, any point on or below this line means there is no different of detecting something then just flip a coin randomly, which means it is useless.

Four study cases with different thresholds were designed and studied which show in the Figure 4.7 for seeking the balance between FPR and TPR. If a low threshold has

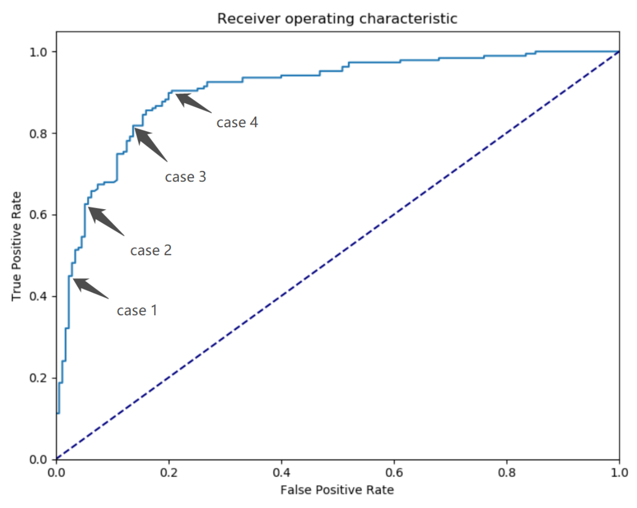


Figure 4.7: Figure illustrates the ROC curve for case study 3

been set, this means the fraction of False positive increases, the model may mispredict some actual failed test cases as passed test cases. Therefore, in order to cover most failed cases, the threshold should be set relatively higher than usual. This can be expressed as following equation, where O is output from the neural network:

$$\text{Class Label} = \begin{cases} \text{Pass} & \text{if} \quad O > \text{Threshold} \\ \text{Fail} & \quad \text{Otherwise} \end{cases} \tag{4.4}$$

Figures 4.8 to 4.11 illustrate the confusion matrices in four different cases. The left figure represents the numerical value and the right figure represents the percentage.

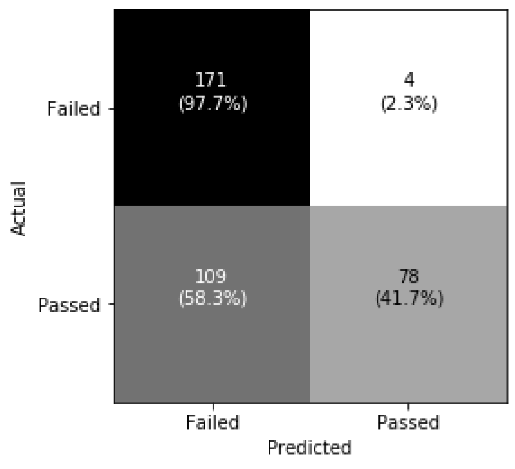


Figure 4.8: Case1, the threshold has been set as 0.862

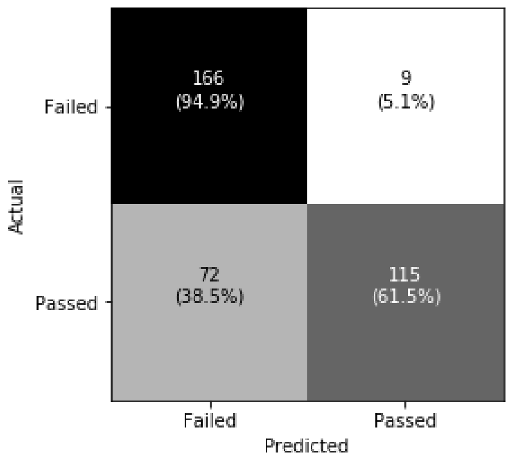


Figure 4.9: Case2, the threshold has been set as 0.822

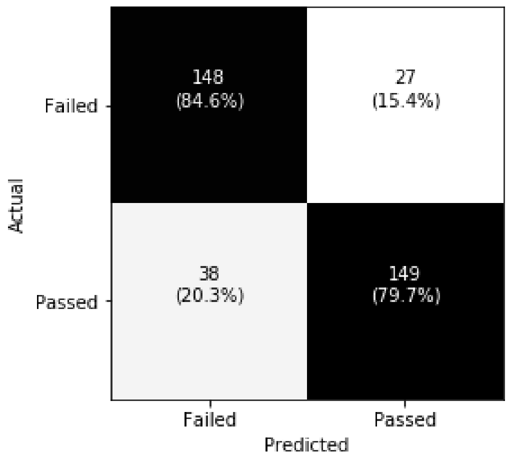


Figure 4.10: Case3, the threshold has been set as 0.67

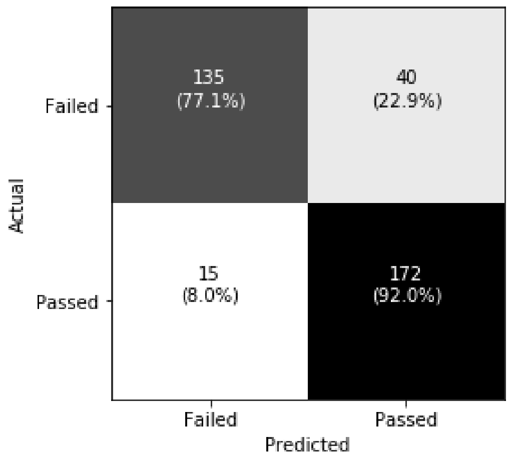


Figure 4.11: Case4, the threshold has been set as 0.24

The investigation which shows above illustrates that with setting relative high threshold around 0.85, the model can predict correctly and cover around 90% accuracy of the actual failed test cases, and still enable to filter out approximately 40% to 60% of passed test cases which means. This means the company can save around half of the time for automated testing.

4.4 Results: GANs for Sequential Data Augmentation

As described in Section 2.2.5.2, LeakGAN has shown promising results in generating long consistency outputs. To examine whether it is suitable and applicable to further improve the classification performance for automating testing. This section describes the outcome of using LeakGAN for generating synthetic log-file as data augmentation. The experimental of model parameter, input data are presented.

4.4.1 Parameter Tuning

Table 4.3 describes the fine-tuned parameter of LeakGAN model. Compare to the original official synthetic data demonstration has up to 10000 amount of data, our data is much smaller (around 900) and much simplified in terms of quantity and complexity accordingly. It is intuitively to reduce the complexity of the model.

In terms of N-gram filter which has been explained in Sections 2.2.5.3 and 2.2.3, with several experiments, we found with either too large or too small filter size leads the model collapses, the loss of discriminator is either fluctuate vastly or very large.

Generator embedding dimension	32
Generator hidden layer	32
Generator learning rate	0.005
Discriminator Embedding dimension	5
Discriminator hidden layer Dimension	32
Discriminator learning rate	0.003
Sequence Length	150
Batch Size	64
Goal dimension size	16
Discriminator filter size	$4, 5, 6, 7, 8, 9, 10, 15, 20$
The number of n-gram filter	200,100,150,150,150,150,150,150,150

Table 4.3: Table describes the fine-tuned hyper-parameter for the model

The learning of generator is equivalent to the standard setting which has been set as 0.005 can obtain best performance. Regarding the learning rate of discriminator, with imperial experiment, we discover with too small learning will cause model collapse and losing diversity, the model performs best when it has been set as 0.003.

4.4.2 Pre-Training

We followed the LeakGan algorithm, pre-train the generator and discriminator interleavingly. With pre-trained module it helps to conduct gradient descent and update generator more effiectively in adversarial training.

4.4.2.1 Generator

The LeakGAN model has two sub-generator including Worker W and Manager M which has described in 2.2.5.2:

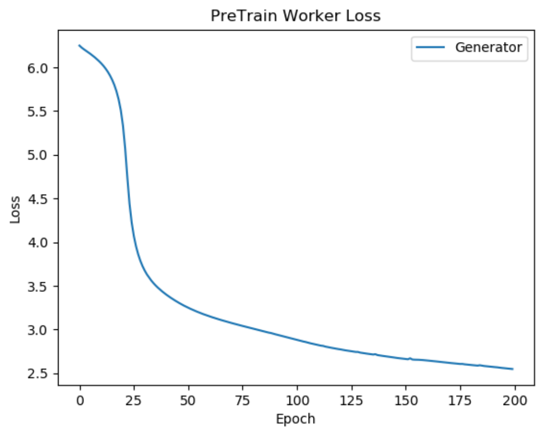


Figure 4.12: Pre-training Worker Loss

Fig 4.12 illustrates the Loss of worker that it converges around 2.5 after 200 epochs.

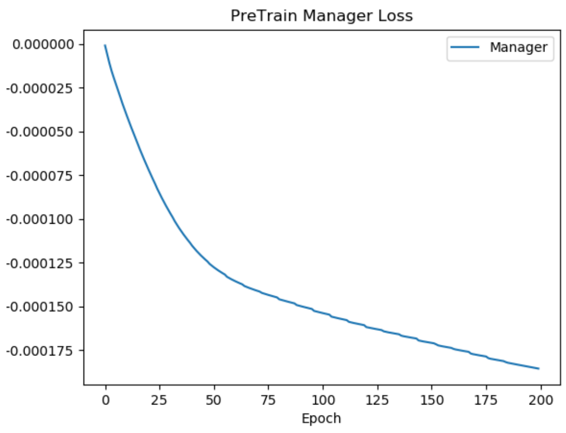


Figure 4.13: Pre-training Manager Loss

And the manager's loss shows in Fig 4.13, since the plot Y-axis is in small-scale, although it keeps decreasing, still it is pretty substantial.

4.4.2.2 Discriminator

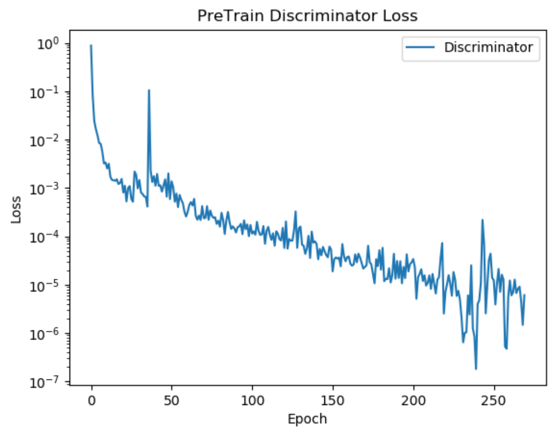


Figure 4.14: Pre-training Discriminator Loss in log-scale

Fig 4.14 shows that the discriminator loss converges to zero since it applies cross entropy as loss function and therefore, it is naturally that it will converge to zeros if Discriminator can accurately distinguish between synthetic data and real data.

4.4.3 Adversarial Training

Figures 4.15 to 4.17 show the plot obtained in the adversarial training stage.

4.4.3.1 Generator

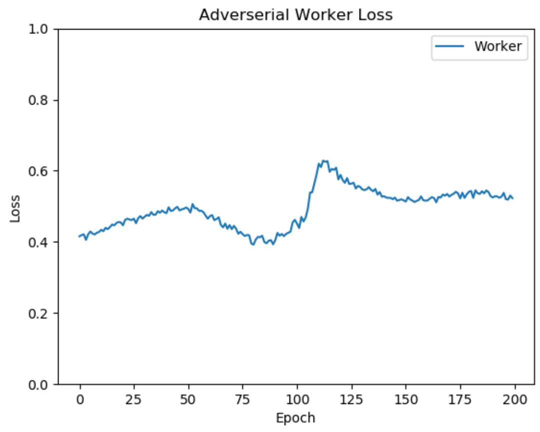


Figure 4.15: Worker Loss

As can see, all the loss are pretty stable and do not have vastly fluctuation, the loss of Manager and Worker does have slightly fluctuation around 100 to 150 epochs, afterwards both loss goes back and maintained around -0.014 and 0.4 accordingly.

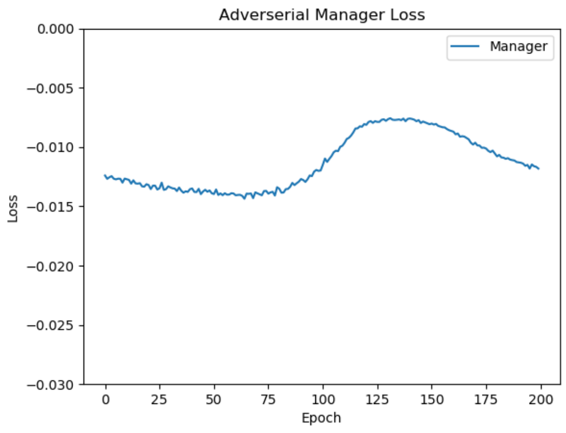


Figure 4.16: Manager Loss in log-scale

If we ignore such small oscillation in the middle of training process. All the figures look pretty stable and do not have enormously decreasing or increasing after 200 epochs.

4.4.3.2 Discrimiantor

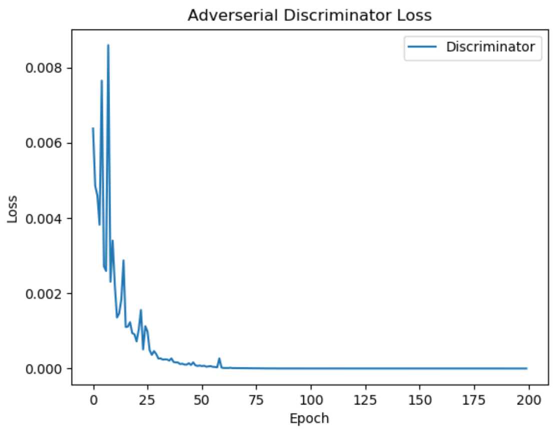


Figure 4.17: Discriminator Loss

The loss of discriminator decreases consistently from 0.008 to 0 after around 50 epoch and does not have any fluctuation after 75 epochs.

4.4.4 Outcome Verification

As discussed in Section 4.1.4 one way to verify whether the outcome from LeakGAN converges to the real data distribution is to check the transfer probability of the dataset. In our case, there are totally 39 test functions and hence, it can be described as a 39 by 39 matrix, Fig 4.18 shows the probability density comparison between the real data and synthetic data.

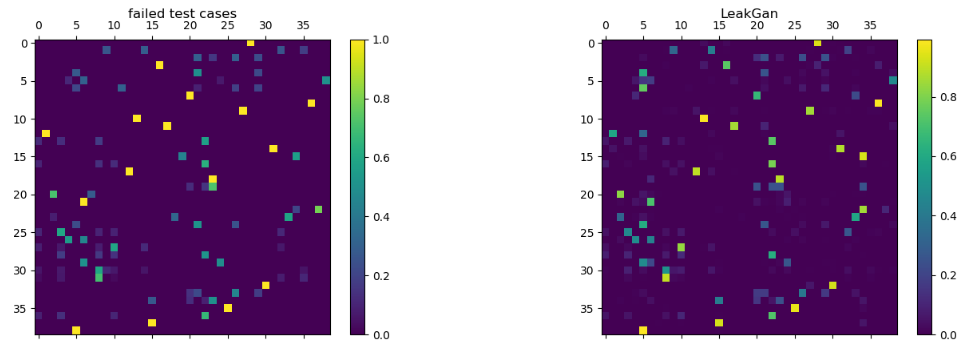
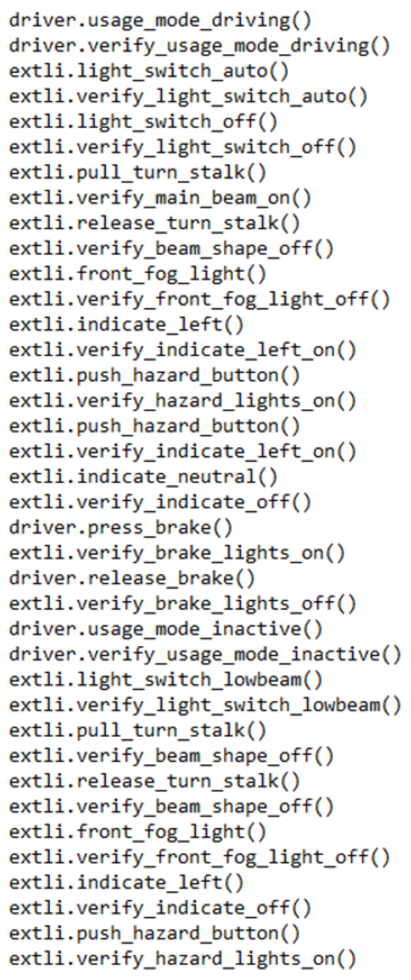


Figure 4.18: Figures above depict the transforming probability distribution. Each line represents the current token, and each column represents the probability from current token to next corresponding token. The color scale shows the probability value from 0 % to 100%,

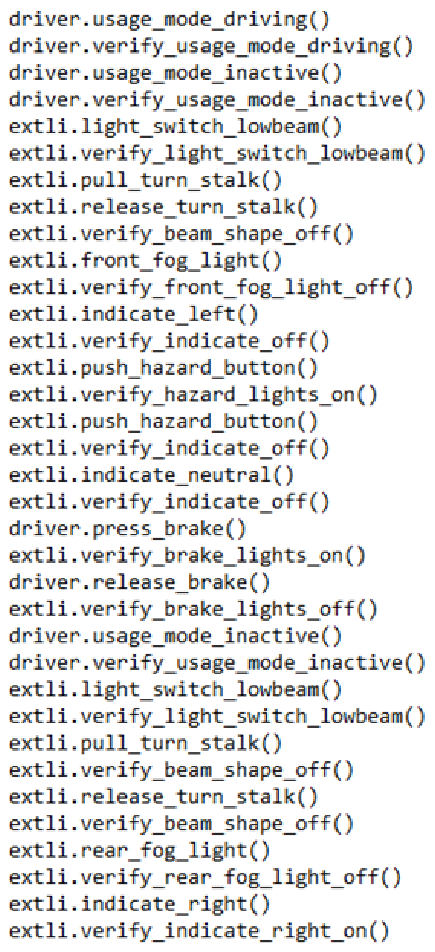
The result indicates that the data has been converged and close to the real data distribution. Although the value of probability transition state for each state is not exactly the same, still the pattern looks pretty similar to the real one. This indicates that the LeakGAN model learns some feature successfully and converges to the real data distribution.

Another alternative way is to let human observers review the outputs to evaluate the quality of generated sequence. Figures 4.19 and 4.20 show the comparison between real data which samples from failed test cases and the synthetic test sequence which generated from LeakGAN. As can see, the synthetic data is pretty similar to the real test case. In the beginning, it follows the topology pattern and randomly chooses some driving mode, after that some function test of external light components begins. However, if we close look these sequence step, the synthetic data sometimes seems do not fulfill with test logic. For instance, after executing extli.rear_fog_light() from last fourth step, the next step should verify if the rear fog light component has been turned on. Conversely, it executes the function to check if it has been turned off.



```
driver.usage_mode_driving()
driver.verify_usage_mode_driving()
extli.light_switch_auto()
extli.verify_light_switch_auto()
extli.light_switch_off()
extli.verify_light_switch_off()
extli.pull_turn_stalk()
extli.verify_main_beam_on()
extli.release_turn_stalk()
extli.verify_beam_shape_off()
extli.front_fog_light()
extli.verify_front_fog_light_off()
extli.indicate_left()
extli.verify_indicate_left_on()
extli.push_hazard_button()
extli.verify_hazard_lights_on()
extli.push_hazard_button()
extli.verify_indicate_left_on()
extli.indicate_neutral()
extli.verify_indicate_off()
driver.press_brake()
extli.verify_brake_lights_on()
driver.release_brake()
extli.verify_brake_lights_off()
driver.usage_mode_inactive()
driver.verify_usage_mode_inactive()
extli.light_switch_lowbeam()
extli.verify_light_switch_lowbeam()
extli.pull_turn_stalk()
extli.verify_beam_shape_off()
extli.release_turn_stalk()
extli.verify_beam_shape_off()
extli.front_fog_light()
extli.verify_front_fog_light_off()
extli.indicate_left()
extli.verify_indicate_off()
extli.push_hazard_button()
extli.verify_hazard_lights_on()
```

Figure 4.19: Real failed test cases.



```
driver.usage_mode_driving()
driver.verify_usage_mode_driving()
driver.usage_mode_inactive()
driver.verify_usage_mode_inactive()
extli.light_switch_lowbeam()
extli.verify_light_switch_lowbeam()
extli.pull_turn_stalk()
extli.release_turn_stalk()
extli.verify_beam_shape_off()
extli.front_fog_light()
extli.verify_front_fog_light_off()
extli.indicate_left()
extli.verify_indicate_off()
extli.push_hazard_button()
extli.verify_hazard_lights_on()
extli.push_hazard_button()
extli.verify_indicate_off()
extli.indicate_neutral()
extli.verify_indicate_off()
driver.press_brake()
extli.verify_brake_lights_on()
driver.release_brake()
extli.verify_brake_lights_off()
driver.usage_mode_inactive()
driver.verify_usage_mode_inactive()
extli.light_switch_lowbeam()
extli.verify_light_switch_lowbeam()
extli.pull_turn_stalk()
extli.verify_beam_shape_off()
extli.release_turn_stalk()
extli.verify_beam_shape_off()
extli.rear_fog_light()
extli.verify_rear_fog_light_off()
extli.indicate_right()
extli.verify_indicate_right_on()
```

Figure 4.20: Sythetic test cases which generated from LeakGAN.

4.4.5 Data Augmentation Assessment

Lastly, to verify whether the augmented data is useful for classification improvement. We use the same model structure as LSTM detection system. The training process has been divided into two steps, the first step is pre-training process, the training data is composed of augmented data which generated from LeakGan and real passed test cases.

# Augmented	# Original DATA	# Total	Training	Validation	Testing
5000	5000	10000	7200	1800	1000
		100%	72%	18%	10%

Table 4.4: Table shows the amount of augmented data sequences which have used to pre-train the neural network and the ratio of training ,validation, and testing data.

Figures 4.21 and 4.22 show that during the pre-training stage, it seems the model

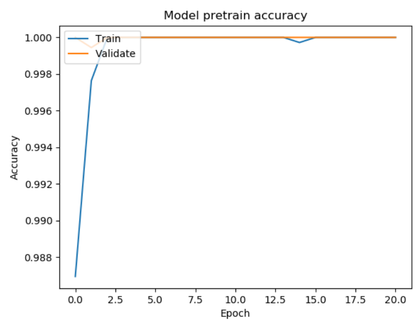


Figure 4.21: Pre-training Accuracy

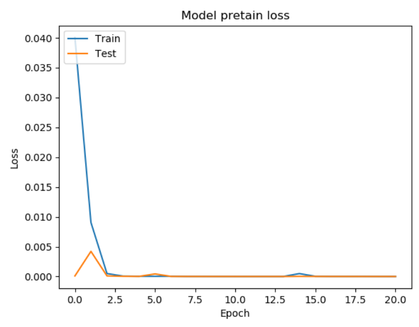


Figure 4.22: Pre-training Loss

can distinguish between synthetic data and real data instantly after two epoch, loss of pertaining process decreases to zero as well and pretty stable after two epochs.

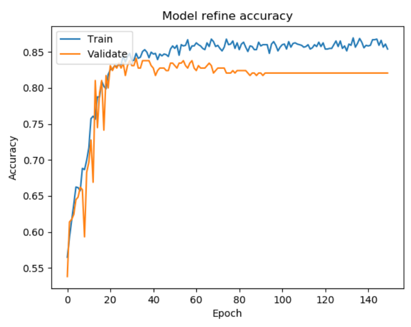


Figure 4.23: Refinement Training accuracy

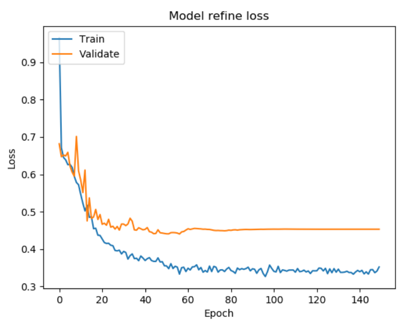


Figure 4.24: Refinement Training Loss

Figures 4.23 and 4.24 illustrate the model performance in refinement training. The hyper-parameter setting has been set as same as novel detection which described in Section 4.3.1. The training accuracy can achieve and stabilize around 86% after 60 epochs. However, the validation accuracy decreases, this indicates the model has slightly over-fitting, which can be observed in loss figure as well.

4.4.6 Discussion

The original purpose is to apply SeqGAN for data-augmentation. Occasionally, we found another paper during the research with same team leader which pointed out that the SeqGAN model suffers from generating long sequence (around 20). Nonetheless, the SeqGAN paper never discussed such issue. LeakGAN has demonstrated excellent result in generating long sequences. However, the research paper has only showed demonstration which sequence has been limit as 40 length. This makes model became uncertainty that whether it is still efficient when sequence is larger than 100 length.

4.5 Final Results Comparison

Two training figs. 4.25 and 4.26 are listed in order to have a intuitive assessment of the data augmentation performance. As can see that both two models can reach approximately 0.85% around 60 epochs on training accuracy and have slightly overfitting issue which are normal and the model performance behavior are pretty identical.

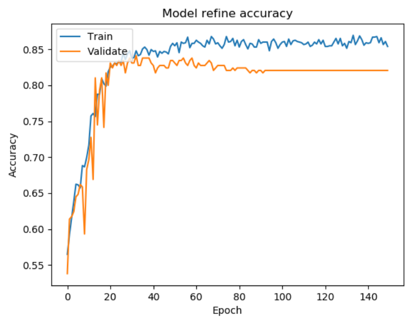


Figure 4.25: Refinement training accuracy with GAN Data Augmentation

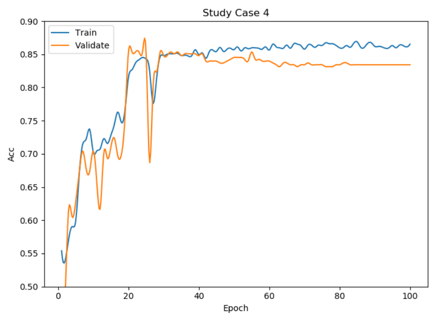


Figure 4.26: Training accuracy without Data Augmentation

Table 4.5 illustrates the performance comparison on test data-set. Notice that a random seed is applied to split the data deterministic randomly, so the test data-set is precisely the same in order to eliminate the assessment bias. The results show that there is no significant improvement with data augmentation.

	Test Accuracy
With GAN Data Augmentation	0.845
Study Case 4	0.850

Table 4.5: Table illustrates the test accuracy comparison between with Data Augmentation and without Data augmentation(study case 4)

4.5.1 Disscussion

The reason for using GAN data augmentation yields worse result is that maybe the LeakGAN model still not converge to real data distribution in spite of the output looks pretty decent, logic and similar as test cases. Since the research paper has demonstrated the experimental data which has been limited as 40 lengths. But in our case, the sequence length is 150 which far exceeded than experimental data. This makes the model suspiciously whether it can conduct gradient descent effectively.

5 Conclusion

In this project, we showed that RNN neural network has a potential application for automating testing based on log-file. The outcomes show that the LSTM detection system can achieve around 86% accuracy on test set with a small amount of training data plus augmented data sequences(by LeakGAN). By setting relative high threshold around 0.8, the model can filter out around 60% of passed test cases but still can distinguish and cover 90% of failed test cases. This means the company can save at least 50% of the time of automating testing but still can discover the majority of test issue.

For LeakGAN, the outcome shows that the model can generate reasonable sequential output with fine-tuned parameter. However, the result shows that there is no improvement for prediction performance by using GANs for data augmentation. Although LeakGAN has shown promising results in generating long sequences, currently it does not satisfy our specific application. Further study is required on this issue.

Future Work

Due to the limit of time, there are still lots of topic can be further researched and studied. Since training leakGAN is pretty time consuming and hence, keep fine-tuning the model to verify if the performance can be further improved.

There are still some other models which has breakthrough in generating discrete sequential data. For instance, Variational Auto Encoder(VAE) is another type generative model which has promising result in NLP field. In term of GAN, RankGAN MaliGAN and TextGAN also show promising result in generation sentences.

Anomaly Detection is another interesting frontier field in Machine learning. One class neural network(OC-NN) proposed by [1] (Chalapathy et al. [2018a]) is the start-of-the-art for outlier detection. The mode has capable to extract feature of one class data and represents a tight envelope of data. Since 95% of our data is passed test cases and therefore, by feeding all the passed into neural network instead of conducting down-sampling might be a suitable topic to conduct research.

References

[1] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Anomaly detection using one class neural networks, 2018.

[2] Christopher Olah. Understanding lstm networks, 2015 August 27,.

[3] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. *CoRR*, abs/1709.08624, 2017.

[4] Advantages of automation, 2018. [online] http://www.softwaretestingmentor.com/advantages-of-automation/.

[5] SAMBHAR PATI. A few useful things to know about machine learning, 2015. [online] https://homes.cs.washington.edu/ pedrod/papers/cacm12.pdf.

[6] Artificial neural networks—mapping the human brain, 2018. [online] https://medium.com/predict/artificial-neural-networks-mapping-the-human-brain-2e0bd4a93160.

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

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

[9] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM neural network for text classification. *CoRR*, abs/1511.08630, 2015.

[10] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[11] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. “generative adversarial networks”. 2014. https://arxiv.org/abs/1406.2661.

[12] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *arXiv e-prints*, page arXiv:1609.05473, Sep 2016.

[13] Christian S. Perone. Machine learning :: Cosine similarity for vector space models (part iii), 2013 September 12. http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/.

[14] Christopher D. Manning Minh-Thang Luong, Richard Socher. Better word representations with recursive neural networks for morphology, 2012.

[15] Lilian Weng. A (long) peek into reinforcement learning, 2018 Feb 19. https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.htmlkey-concepts.

[16] Thomas Simonini. An introduction to policy gradients with cartpole and doom, 2018 May 9. https://medium.freecodecamp.org/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f.

[17] B. Mehlig. Artificial neural networks. abs/1506.00019, January 2019.

[18] Alex Krizhevsky Ilya Sutskever Ruslan Salakhutdinov Nitish Srivastava, Geoffrey Hinton. Dropout: A simple way to prevent neural networks from overfitting, 2014.

[19] D.Lanzo A.Palladino, G.Fiengo. a-portable-hardware-intheloop-device-for-automotive-diagnostic-control-systems, 2010 April 22. https://www.slideshare.net/isa$_interchange/a - portable - hardware - intheloop - device - for - automotive - diagnostic - con$.